POLITECHNIKA KRAKOWSKA
IM. TADEUSZA KOŚCIUSZKI
WYDZIAŁ FIZYKI MATEMATYKI I INFORMATYKI
KIERUNEK INFORMATYKA

JAKUB CZYRZEWSKI

NOWE CECHY, FUNKCJE ORAZ MECHANIZMY DOSTĘPNE W ORACLE DATABASE 10G I 11G

NEW FEATURES OF ORACLE DATABASE 10G AND 11G

PRACA INŻYNIERSKA
STUDIA STACJONARNE

Promotor: dr inż. Stanisława Plichta

Kraków 2012

Spis treści

1. Wstęp 4

1.1. Historia baz danych 4

1.2. Cel i zakres pracy 7

2. Instalacja i konfiguracja środowiska testowego 9

2.1. Koncepcja 9

2.2. Przygotowanie wirtualnego systemu operacyjnego 10

2.3. Instalacja oprogramowania SZBD 11

2.4. Utworzenie *listenera* bazy danych 13

2.5. Utworzenie bazy danych 14

3. Aplikacja testowa 17

3.1. Opis poglądowy 17

3.2. Masowe przetwarzanie i ładowanie 18

3.2.1. Gromadzenie danych 19

3.2.2. Analiza składniowa 21

3.2.3. Ładowanie do bazy danych 22

3.3. Schemat bazy danych 23

3.4. Przygotowanie zbioru danych 27

4. Nowe funkcjonalności 28

4.1. Wirtualne kolumny 28

4.2. Operatory transformacji 31

4.2.1. Operator *PIVOT* 31

4.2.1. Operator *PIVOT* z opcją *XML* 35

4.2.2. Operator *UNPIVOT* 36

4.3. Usprawnienia w instrukcjach *DDL* 38

4.3.1. Ulepszone ograniczenie *NOT NULL* 38

4.3.2. Automatyczne oczekiwanie na zdjęcie blokady 39

4.3.3. Tabele tylko do odczytu 40

4.3.4. Niewidzialny indeks 41

4.4. Zarządzanie statystykami 43

4.4.1. Stany statystyk 43
4.4.2. Publikacja statystyk 44
4.4.3. Repozytorium starych statystyk 45
4.4.4. Przywracanie statystyk 46
4.5. Partycjonowanie 47
4.5.1. Partycjonowanie przedziałowe 48
4.5.2. Partycjonowanie złożone 52
4.5.3. Partycjonowanie referencyjne 57
4.5.4. Partycjonowanie systemowe 61
4.6. Wyzwalacze 62
4.6.1. Wyzwalacze złożone 62
4.6.2. Sterowanie porządkiem wykonania 65
4.7. Kompresja tabeli 66
4.8. Technologia *Flashback* 71
4.8.1. Dostosowanie instancji bazy danych 72
4.8.2. Flashback Database 74
4.8.3. Flashback Drop 77
4.8.4. Flashback Table 77
4.8.5. Flashback Version Query 78
4.8.1. Flashback Transaction Query 80
4.8.2. Flashback Transaction 80
4.8.3. Flashback Data Archive 85

5. Dostępność nowych funkcjonalności 88

6. Podsumowanie 89

7. Literatura 93

8. Spis tabel 95

9. Spis rysunków 97

1. Wstęp

1.1. Historia baz danych

Bazy danych towarzyszą ludzkości niemal od samego początku istnienia. Pierwotnie dane były gromadzone i przetwarzane ręcznie w formie glinianych tabliczek, papirusów oraz papieru. Taki stan rzeczy utrzymywał się bardzo długo. Pomimo wynalezienia druku oraz urządzeń do szybkiego kopiowania, dane ciągle były przetwarzane ręcznie. Dynamiczny rozwój, spowodowany głównie rewolucją przemysłową, wymusił poddanie automatyzacji procesu gromadzenia i przetwarzania danych. Pozwoliło na to pojawienie się kart perforowanych oraz urządzeń mechanicznych
i mechaniczno-elektrycznych. Nowe rozwiązanie przyjęło się bardzo szybko, ponieważ pozwalało na przetwarzanie ogromnej ilości informacji. Już wtedy nie było możliwości zastąpienia maszyn przez rzeszę ludzi. Pozwalały one skrócić ośmiokrotnie czas opracowania wyników spisu powszechnego, który odbył się pod koniec XIV wieku w Stanach Zjednoczonych.

Największy rozwój baz danych przypada na ostatnie 60 lat. Powodem tego było zbudowanie pierwszych komputerów oraz wynalezienie taśm magnetycznych. Komputery pozwalały na szybsze i łatwiejsze dostosowanie ich do wykonywania niestandardowych zadań. Opracowano pierwsze prymitywne odmiany baz danych w postaci systemów plików, które zarządzają danymi gromadzonymi na nośniku. Pomimo ogromnego postępu, ówczesne rozwiązania nadal posiadały wady. Były to wymuszone sekwencyjne przetwarzanie danych oraz brak interakcji spowodowany przetwarzaniem wsadowym [1]. Problemy te zostały rozwiązane poprzez zaprojektowanie dysku magnetycznego, który pozwala na swobodny dostęp do danych oraz urządzeń *I/O*. Wraz z nową koncepcją dostępu do danych, zaimplementowano bardziej zaawansowane systemy plików, które pozwalały na odczyt i zapis dowolnych fragmentów plików z obszaru nośnika. Był to mechanizm, który stał się prekursorem dzisiejszych baz danych i może zostać określony terminem „płaska baza danych”.

Pierwszym modelem bazy danych był model hierarchiczny. Dane zorganizowane były w formie drzewa. Na każdym poziomie znajdował się inny typ rekordu. Hierarchia taka wywodzi się bezpośrednio z systemu plików, ponieważ przypomina strukturę

katalogów na nośniku danych. Wprowadzono centralny mechanizm składowania i zarządzania danymi. Twórcy systemów informatycznych, korzystających z bazy danych, zostali zwolnieni z implementacji metod służących do odczytu, modyfikowania, dodawania i usuwania. Była również zapewniona współbieżna praca wielu końcowych użytkowników. Operacje te nie są proste w implementacji oraz wymagają ogromnej wiedzy i nakładów pracy. Zastosowanie modelu hierarchicznego pozwoliło na redukcję kosztów i czasu tworzenia oprogramowania, przy jednoczesnym zwiększeniu bezpieczeństwa i niezawodności. Pomimo korzyści, model posiadał wady. Naturalnymi dostępnymi relacjami pomiędzy rekordami są „jeden do wielu" i „wiele do jednego". Wynika to bezpośrednio z warunków integralności danych, które mówią, że każdy rekord może mieć tylko jednego rodzica. W przypadku realizacji relacji „wiele do wiele" należało kopiować całą gałąź danych do innego węzła macierzystego. Wymuszona redundancja powodowała anomalię wstawiania i aktualizacji. Ponadto usunięcie rekordu nadrzędnego powodowało utratę całej gałęzi drzewa, znajdującej się pod usuwanym rekordem (anomalia usuwania).

Rozwinięciem modelu hierarchicznego był model sieciowy. Eliminował on wady poprzednika, wynikające z warunków integralności. Hierarchia drzewiasta została zastąpiona grafem. Każdy rekord mógł mieć przypisanych kilku potomków i rodziców. Przeglądanie danych polegało na przechodzeniu pomiędzy rekordami korzystając z łączy. W czasach dominacji modelu sieciowego rozpoczęto pracę nad standardami dotykającymi zagadnień baz danych. Opracowano standardy języków *DDL* i *DML*, które służą odpowiednio do definiowania struktur logicznych bazy danych oraz manipulowania (dodawanie, usuwanie, modyfikacja) danymi. Zaproponowano koncepcję, która wyróżniała trzy rodzaje schematów – logiczny, fizyczny oraz podschemat. Dzięki takiemu podziałowi udało się odseparować rozwój baz danych od rozwoju aplikacji z nich korzystających. Podschematy pozwalały na zwiększenie bezpieczeństwa poprzez udostępnienie jedynie fragmentu bazy danych na potrzeby aplikacji. Wprowadzono także pojęcie transakcji, która pozwalała uniknąć anomalii w przypadku współbieżnego dostępu do danych. Dziennik zmian umożliwiał wycofacie wszystkich wykonanych wcześniej operacji, w przypadku niepowodzenia bieżącej instrukcji transakcji. W trakcie eksploatacji baz danych modelu sieciowego zdiagnozowano wady. Jedną z nich był skomplikowany proces nawigacji po rekordach. Nierzadko prosta zmiana struktury bazy

danych wymuszała przebudowę wszystkich zapytań wykorzystywanych w systemie informatycznym, ponieważ w ich definicji znajdowała się pełna ścieżka dostępu. Pomimo tej wady, modele sieciowe dominowały aż do końca lat 80. XX wieku.

Zupełną innowacją okazał się model relacyjny. Wprowadził on nową reprezentację danych w postaci zbiorów rekordów tej samej encji, zwanych relacjami. Związki między rekordami są realizowane niejawnie poprzez dodatkowe atrybuty w encjach będących w relacji ze sobą. Pozwala to na zastosowanie prostych operatorów algebraicznych do manipulacji zbiorami. Zapytania zwracające rekordy z bazy danych zmieniły swój charakter, opisując jedynie cechy zbioru wynikowego zamiast pełnej ścieżki dostępu do danych. Niskopoziomowym dostępem do zbiorów oraz operacjami, jakie należy wykonać w celu otrzymania żądanego rezultatu, zajmuje się system zarządzania bazą danych. Pozwala to na łatwiejsze utrzymanie systemu. Tylko gruntowane zmiany w strukturze bazy danych pociągają za sobą zmiany w zapytaniach wykorzystywanych przez aplikację. Specjalnie dla modelu relacyjnego opracowano język *SQL*, zgodny ze standardami *DDL* i *DML*. Powstało również pojęcie normalizacji bazy danych, pod którym kryją się zasady i reguły, jakie należy wykonać, aby zminimalizować występującą redundancję oraz zapobiec występowaniu anomalii. W zakres odpowiedzialności SZBD weszła optymalizacja ścieżki dostępu do danych za pomocą optymalizatora oraz buforowanie, które pozwala na redukcję operacji *I/O* wykonywanych przez podsystem dyskowy. Na początku lat 90. XX wieku model relacyjny rozpoczął okres swojej dominacji, który trwa do dziś. Każdy kolejny model bazy danych posiadał wady. Nie inaczej jest w przypadku baz relacyjnych. Najpoważniejszą z nich jest brak typów złożonych, który wynika z zasad normalizacji mówiących o tym, że każdy atrybut tabeli musi być atomowy. Ponadto operacje wykonywane na encjach są całkowicie odseparowane od danych. Wady te częściowo wyeliminowano poprzez dodanie nowych typów oraz umożliwienie tworzenia własnych typów wraz z zestawem operacji.

Wady modelu relacyjnego spowodowały rozpoczęcie prac nad modelem obiektowym, który enkapsuluje rekord oraz metody, służącego do manipulacji tymże rekordem. Pozwala na definiowanie własnych typów przez programistów oraz umieszczanie ich w tabelach obiektowych. Model ten nie przyjął się i nie zdominował rynku jak jego poprzednik, ze względu na uzupełnienie modelu relacyjnego o cechy

systemów obiektowych. Jako rezultat rywalizacji, dwóch konkurencyjnych koncepcji, powstał model relacyjno-obiektowy.

Obecnie bazy danych są powszechnie stosowane w każdej dziedzinie życia oraz łatwo dostępne. Istnieje wiele SZBD, darmowych jak i komercyjnych, posiadających zróżnicowane funkcjonalności. Dominuje model relacyjno-obiektowy. Elementy obiektowości są najczęściej pomijane przy projektowaniu bazy danych, ale nadal dostępne dla twórców oprogramowania. Bardzo dobrym pomysłem okazało się wprowadzenie przez *Oracle Corporation* widoków obiektowych. Wraz z wyzwalaczami typu *INSTEAD OF*, pozwalają one na zaadoptowanie bazy danych zbudowanej w konwencji relacyjnej na potrzeby koncepcji obiektowej [2]. Dzięki takiemu widokowi programista ma wrażenie, że korzysta z modelu w pełni obiektowego. Relacyjność bazy danych staje się dla niego przeźroczysta, ponieważ manipuluje jedynie obiektami.

1.2. Cel i zakres pracy

Celem niniejszej pracy jest przegląd cech, mechanizmów oraz funkcji *Oracle Database 10g* i *11g* oraz ocena przydatności, wydajności i wygody użytkowania. Nowa funkcjonalność zostanie wykorzystana w kontekście aplikacji testowej. Realizacja celu umożliwi poszerzenie wiedzy o środowisku bazodanowym. Jego dobra znajomość pozwala na optymalizację działania aplikacji oraz oparcie implementacji nowych funkcjonalności na sprawdzonych rozwiązaniach dostarczonych przez *Oracle Corporation*. W skrajnych przypadkach, konieczne może stać się przerobienie istniejącej funkcjonalności, tak by aktywnie wykorzystywała generyczne mechanizmy bazy danych, w celu osiągnięcia lepszej skalowalności, wydajności i łatwości utrzymania.

Praca została podzielona na trzy zasadnicze etapy, które wchodzą w jej zakres. Pierwszy etap to uruchomienie środowiska testowego składającego się z pojedynczej instancji bazy danych. Użyta zostanie najnowsza wersja SZBD tj. *Oracle Database 11g Release 2.* Drugi dotyczy projektu i implementacji schematu bazy danych oraz aplikacji, które pozwolą w łatwy sposób masowo ładować informacje do tabel. Trzecim, ostatnim etapem jest wykorzystanie narzędzi, które zostały skonfigurowane i zaimplementowane w poprzednich etapach, do praktycznego przetestowania wybranych mechanizmów i funkcjonalności oraz zbadania ich cech. Literatura podaje bogaty zbiór tychże [3] [4]

[5] [6] [7]. Na podstawie ostatniego etapu, możliwa staje się ocena przydatności, wydajności
i wygody użytkowania.

2. Instalacja i konfiguracja środowiska testowego

2.1. Koncepcja

Testowa instancja bazy danych została uruchomiona w kontekście systemu wirtualnego. Jako środowisko dla wirtualnej maszyny, wykorzystany został program *Oracle VM VirtualBox* w wersji 4.2.4. Rolę wirtualnego systemu operacyjnego pełni *Oracle Linux 6.3* z jądrem w wersji 2.6.39. System ten zapewnia pełne wsparcie dla bazodanowych produktów firmy *Oracle*. Maszyna wirtualna posiada do dyspozycji 4 GB *RAM*, 2 rdzenie procesora, dwie wirtualne karty sieciowe oraz wirtualny dysk o wielkości 50 GB. W tabeli 1 znajduje się rozkład woluminów na tymże dysku. Przy podziale, skorzystano z *Logical Volume Manager*. Pozwoli to na późniejsze dostrojenie wielkości woluminów, w zależności od realnych potrzeb [8]. Strukturę zaprojektowano zgodnie ze standardem *Optimal Flexibile Architecture,* który opisuje hierarchię katalogów oprogramowania *Oracle* [9]. Punkty montowania */u01*, */u02* oraz */u03* przechowują odpowiednio oprogramowanie, dane oraz obszar odzyskiwania. Interfejs sieciowy *eth0* został skonfigurowany w trybie *Bridged Adapter*, dzięki czemu będzie otrzymywał od serwera *DHCP* unikalny adres *IP* w sieci lokalnej. Interfejs *eth1* został skonfigurowany w trybie *Host-only Adapter*. Pozwoli to na korzystanie z serwera bazy danych, na rzeczywistym systemie hosta, w przypadku braku dostępu do sieci.

Urządzenie	**Wielkość**	**Punkt montowania**	**System plików**
/dev/sda1	1GB	/boot	ext4
/dev/mapper/testdbds-root	7GB	/	ext4
/dev/mapper/testdbds-swap	4GB	-	swap
/dev/mapper/testdbds-tmp	3GB	/tmp	ext4
/dev/mapper/testdbds-var	2GB	/var	ext4
/dev/mapper/testdbds-home	2GB	/home	ext4
/dev/mapper/testdbds-orasoft	8GB	/u01	ext4
/dev/mapper/testdbds-orafra	5GB	/u03	ext4
/dev/mapper/testdbds-oradata	5GB	/u02	ext4

Tabela 1. Rozkład woluminów na wirtualnym dysku

Maszyna rzeczywista, będąca hostem dla wirtualnego systemu, działa pod kontrolą systemu operacyjnego *GNU/Linux Debian 6.0.6*. W tabeli 2 znajduje się jej

specyfikacja. Wydajność pamięci masowej została zbadana za pomocą programu *Crystal Disk Mark* w wersji 3.0.2. Podany wynik dotyczy sekwencyjnego odczytu i zapisu przy wielkości próbki 1000 MB.

Parametr		**Nazwa/Wartość**
Procesor		Intel® Core™ i7-3610QM
Pamięć operacyjna		16 GB DDR3 1600 MHz
Pamięć masowa		VTX4-25SAT3-256G
Wydajność pamięci masowej	Odczyt	402 MB/s
	Zapis	427 MB/s

Tabela 2. Specyfikacja rzeczywistej maszyny hosta.

2.2. Przygotowanie wirtualnego systemu operacyjnego

Po pomyślnym przeprowadzeniu instalacji bazowego systemu, wykonano pełną aktualizację.

```
[root@testdbds ds]# yum update
```

Następnie zainstalowano pakiet *oracle-rdbms-server-11gR2-preinstall*, który pozwala na pełne przygotowanie systemu operacyjnego do instalacji oprogramowania bazy danych *Oracle Database 11g Release 2*. W przypadku nieskorzystania z udogodnienia, należałoby wykonać szereg operacji samodzielnie. W dokumentacji technicznej znajduje się pełny opis wymagań [9].

```
[root@testdbds oracle]# yum install oracle-rdbms-server-11gR2-
preinstall
```

Do pliku *.bash_profile* znajdującego się w katalogu domowym użytkownika *oracle*, dodano definicję poniższych zmiennych środowiskowych.

```
#Bazowa lokalizacja oprogramowania Oracle
ORACLE_BASE=/u01/app/oracle; export ORACLE_BASE
#Domowy katalog oprogramowania bazy danych
ORACLE_HOME=$ORACLE_BASE/product/11.2.0/dbhome_1; export ORACLE_HOME
#Identyfikator usługi
ORACLE_SID=testdbds; export ORACLE_SID
#Graficzny typ terminalu
ORACLE_TERM=xterm; export ORACLE_TERM
#Unikalna nazwa węzła, istotne przypadku RAC
```

```
ORACLE_UNQNAME=testdbds; export ORACLE_UNQNAME
#Wzbogacenie ścieżki wyszukiwania plików wykonywalnych
PATH=$ORACLE_HOME/bin:$PATH; export PATH
#Ustawienie hosta dla terminala graficznego
DISPLAY=localhost:0.0; export DISPLAY
#Wzbogacenie ścieżki wyszukiwania bibliotek
LD_LIBRARY_PATH=$ORACLE_HOME/lib:/lib:/usr/lib; export LD_LIBRARY_PATH
#Wzbogacenie ścieżki wyszukiwania klas Java
CLASSPATH=$ORACLE_HOME/JRE:$ORACLE_HOME/jlib:$ORACLE_HOME/rdbms/jlib;
export CLASSPATH
#Tryb tworzenia plików
umask 022
```

Aby umożliwić komunikację klientów z oprogramowaniem *Oracle* odblokowano ruch przychodzący na następujących portach dla protokołu *TCP*, w konfiguracji zapory przeciwogniowej:

- 1521 – *Listener* bazy danych
- 1158 – *Oracle Enterprise Manager*

Na tym etapie wykonano kopię zapasową obrazu systemu. Tak przygotowany *Oracle Linux* jest gotowy do zainstalowania oprogramowania bazy danych dla pojedynczej instancji. Ponadto, za pomocą programu *VirtualBox* utworzono wirtualny dysk współdzielony o wielkości 5 GB. Posłuży on do wygodnego przenoszenia plików instalacyjnych pomiędzy maszynami wirtualnymi. Pobrano, zapisano i rozpakowano na nim oprogramowanie *Oracle Database 11g Release 2* dedykowane dla architektury x86-64 oraz systemu *Linux*.

2.3. Instalacja oprogramowania SZBD

W przypadku prawidłowo skonfigurowanego systemu operacyjnego, instalacja oprogramowania pojedynczej instancji bazy danych przebiega sprawnie i bezproblemowo. Do jej przeprowadzenia należy użyć programu *Oracle Universal Installer*. W trakcie uruchomienia sprawdzana jest podstawowa konfiguracja. Jeśli środowisko graficzne nie spełnia parametrów, instalator nie uruchomi się. W przypadku defektów konfiguracji sieciowej, zostanie wyświetlony alert z możliwością zignorowania problemu.

Po prawidłowym zalogowaniu się na konto użytkownika *oracle* zmieniono użytkownika na *root* i umożliwiono skorzystanie z serwera interfejsu graficznego.

```
[root@testdbds ds]# xhost +localhost
```

Zamontowano współdzielony dysk w katalogu */shareable* a następnie uruchomiono *Oracle Universal Installer* z poziomu użytkownika *oracle*.

```
[oracle@testdbds linux.x64_11gR2_database]$ ./runInstaller
```

Po zakończeniu inicjalizacji instalatora, pojawiło się okno główne. Instalacja przebiega wieloetapowo. Przejście pomiędzy kolejnymi etapami następuje poprzez naciśnięcie przycisku „*Next*". Aby zachować czytelność oraz odtwarzalność procesu, został on przedstawiony w tabeli 3.

Nazwa etapu	**Wykonane czynności**
Configure Security Updates	Odznaczono opcję „I *wish to receive security updates via My Oracle Support*" i naciśnięto przycisk „*Next*". Pojawiło się okno alertu dotyczące wsparcia produktu, w którym wybrano „*Yes*". Spowodowało to kontynuację instalacji.
Installation Option	Wybrano opcję „*Install database software only*". Na etapie instalacji oprogramowania nie warto tworzyć bazę danych.
Grid Options	Zdefiniowano środowisko, w jakim będzie pracować instancja. W tym przypadku jest to „*Single instance database installation*„, co oznacza pojedynczą instancję.
Product Languages	Opcję dostępnych języków pozostawiono bez zmian. Domyślnie instalowana jest wersja angielska.
Database Edition	Ustalono wersję bazy danych na „*Enterprise Edition*". Następnie, w tym samym oknie można zdefiniować, jakie komponenty bazy danych zostaną zainstalowane poprzez naciśnięcie przycisku „*Select Options…*". W widoku, który pojawia się, zaznaczono „*Oracle Database Vault option*" oraz „*Oracle Label Security*".
Installation Location	Istnieje możliwość zdefiniowania katalogu bazowego oprogramowania Oracle. Wartość ta jest ustawiona na *$ORACLE_BASE,* czyli „*/u01/app/oracle/*". Pozostawiono ją bez zmian, ponieważ jest prawidłowa. Pole „*Software Location*", które definiuje katalog domowy oprogramowania bazy danych, również jest poprawnie uzupełnione.
Create Inventory	Istnieje możliwość ustawienia pola „*Inventory Directory*" oraz „*oraInventory Group Name*". Wartości pozostawiono bez zmian, czyli odpowiednio „*/u01/app/oracle/oraInventory*" oraz „*oinstall".*
Operating System Groups	Zdefiniowano, jakie grupy systemu operacyjnego, mają być wykorzystywane przez instancję bazy danych do podziału odpowiedzialności. Jako „*Database Administrator (OSDBA) Group*" wybrano „*dba*" a jako „*Database Operator (OSOPER) Group*" ustawiono grupę „*oinstall*".

Nazwa etapu	Wykonane czynności
Prerequisite checks	Na tym etapie została sprawdzona konfiguracja systemu. *OUI* zweryfikował czy w systemie są zainstalowane dokładnie te wersje pakietów, które zostały wymienione w dokumentacji instalacyjnej [9]. System zawiera nowsze wersje, co jest również dozwolone przez dokumentację. Instalator nie bierze tego pod uwagę, więc zignorowano ostrzeżenia. Konfiguracja jest prawidłowa.
Summary	Zapisano plik raportu, który przechowuje wszystkie parametry instalacyjne, za pomocą przycisku „*Save Response File…*”. Następnie naciśnięto przycisk „*Finish*”.
Install product	Instalacja oprogramowania instancji bazy danych rozpoczęła się. Na samym końcu instalacji uruchomiono dwa skrypty z poziomu użytkownika systemowego *root*: ▪ /oracle/instance/oraInventory/orainstRoot.sh ▪ /oracle/instance/app/oracle/product/11.2.0/db/root.sh
Finish	Naciśnięto przycisk „*Close*”. Instalator zakończył działanie. Oprogramowanie bazy danych zostało prawidłowo zainstalowane.

Tabela 3. Przebieg instalacji oprogramowania bazy danych.

2.4. Utworzenie *listenera* bazy danych

Aby możliwe było zainstalowanie aplikacji *Oracle Enterprise Manager*, w trakcie tworzenia bazy danych, wymagany jest działający *listener*. Pozwala on klientom na podłączenie się do bazy danych. Do zarządzania konfiguracją sieciową służy narzędzie *Network Configuration Assistant*, które zostało uruchomione z poziomu terminalu za pomocą poniższego polecenia.

```
[oracle@testdbds oracle]$ netca
```

Pojawiło się główne okno programu. Proces tworzenia jest wieloetapowy. Przejście pomiędzy kolejnymi etapami następuje poprzez naciśnięcie przycisku „*Next*”. Przebieg został przedstawiony w tabeli 4.

Nazwa etapu	Wykonane czynności
Welcome	Standardowo wybrana opcja to „*Listener configuration*”. Pozostawiono ją bez zmian.
Listener Configuration	W tymże etapie istnieje możliwość zdefiniowania akcji użytkownika. *Listener* może zostać dodany, usunięty lub zmodyfikowany. Domyślnie wybraną opcją jest „*Add*”. Pozostawiono ją bez zmian, ponieważ właśnie ta akcja jest pożądana.
Listener Name	W polu „*Listener name*” wpisano „*testdbdslsnr*”.

Nazwa etapu	Wykonane czynności
Select Protocols	Następuje wybór protokołów sieciowych, jakie będzie akceptował aktualnie definiowany *listener*. Domyślnie wybrany jest jedynie protokół *TCP*. Nie wprowadzono żadnych zmian.
TCP/IP Protocol	Port nasłuchu pozostawiono bez zmian. Standardowo, *listener* nasłuchuje na porcie 1521.
More Listeners?	Oprogramowanie zapytało użytkownika, czy ma zostać wykonana jakaś inna akcja w tej sekcji programu. Pozostawiono wybraną opcję „*No*”.
Select Listener	Pole „*Select listener you wan to start*” wskazuje *listener*, który ma zostać uruchomiony. Domyślnie, zawiera nazwę *listenera*, który został przed chwilą utworzony.
Listener Configuration Done	Program wyświetlił informację o prawidłowym skonfigurowaniu. Żadna akcja nie jest wymagana.
Welcome	Nastąpił powrót do okna powitalnego. Naciśnięto przycisk „*Finish*”, aby zakończyć działanie oprogramowania.

Tabela 4. Przebieg procedury tworzenia *listenera*.

2.5. Utworzenie bazy danych

Do zarządzania bazami danych służy narzędzie *Database Configuration Assistant*. Umożliwia ono tworzenie, usuwanie, rekonfigurację oraz modyfikację szablonów. Uruchomiono je za pomocą poniższego polecenia.

```
[oracle@testdbds oracle]$ dbca
```

Pojawiło się główne okno programu. Proces tworzenia jest wieloetapowy. Przejście pomiędzy kolejnymi etapami następuje poprzez naciśnięcie przycisku „*Next*”. Przebieg został przedstawiony w tabeli 5.

Nazwa etapu	Wykonane czynności
Welcome	Pojawił się ekran powitalny oprogramowania.
Operations	Opcja pozwala na wybór akcji, jaką chce podjąć użytkownik. Pozostawiono domyślną opcję tj. „*Create a Database*”.

Nazwa etapu	Wykonane czynności
Database Templates	W tymże etapie istnieje możliwość skorzystania z predefiniowanych szablonów bazy danych. Nie jest to dobre rozwiązanie, ponieważ zawierają one schematy użytkowników demonstracyjnych. Wybrano opcję „*Custom Database*", która pozwala na zdefiniowanie szczegółowych informacji o fizycznej strukturze bazy danych.
Database Identification	Ustawiono wartość pola „*Global database name*" na „*testdbds"*. Zawartość pola „*Oracle Service Identifier (SID)*" została automatycznie sprowadzona do tej samej wartości.
Management Options	Wszystkie opcje pozostawiono bez zmian. Dzięki temu, że istnieje *listener*, opcja *Configure Enterprise Manager* mogła pozostać załączona.
Database Credentials	Zaznaczono opcję „*Use the Same Administrative Passwords for All Accounts*". Wpisano zbiorcze hasło dla wszystkich wymienionych kont bazodanowych a następnie potwierdzono je.
Database Files Locations	Zaznaczono opcję „*Use Common Location for All Database Files*" i ustawiono lokalizację „*/u02/app/oracle/oradata/*" jako magazyn plików danych.
Recovery Configuration	Pozostawiono standardowo ustawioną maksymalną wielkość obszaru odzyskiwania, który jest wykorzystywany przez technologię *Flashback Database*. Lokalizacją *Flash Recovery Area* został katalog „*/u03/app/oracle/flash_recovery_area/*".
Database Content	Zaznaczono opcję „*Oracle Database Vault*".
Oracle Database Vault Credentials	Ustawiono wartość pola „*Database Vault Owner*" na „*dbvwon*", po czym wprowadzono hasło dla konta i potwierdzono je. Zaznaczono opcję „*Create Separate Account Manager*". W pole „*Database Vault Account Manager*" wpisano „*dbvam*". Następnie wprowadzono hasło dla drugiego konta i potwierdzono je.
Initialization Parameters	W zakładce „*Character Sets*" wybrano kodowanie UTF-8 poprzez zaznaczenie opcji „*Use Unicode (AL32UTF-8)*".

Nazwa etapu	Wykonane czynności
Creation Options	Następuje definicja się newralgicznych elementów bazy danych. Standardowa konfiguracja plików kontrolnych zakłada, że jest on przechowywany w dwóch kopiach: ▪ */u02/app/Oracle/oradata/ testdbds/*, ▪ */u03/app/oracle/flash_recovery_area/ testdbds/*; Jest ona wystarczająca na potrzeby bazy testowej. Zmieniono nazwę przestrzeni tabel wycofania z „*UNDOTBS1*" na „*UNDO*". Ponadto utworzono dwie nowe przestrzenie tabel przeznaczone specjalnie dla aplikacji testowej: ▪ *OIMSTABLE* – segmenty tabel, ▪ *OIMSINDEX* –segmenty indeksów; Domyślna konfiguracja zawiera trzy grupy plików dziennika powtórzeń, które są zapisywane cyklicznie przez proces tła *LGWR*. Nie są jednak tworzone kopie pliku w obrębie grupy. W celu zabezpieczenia plików dziennika powtórzeń, w każdej grupie utworzono jedną kopię przechowywaną w lokalizacji */u03/app/oracle/flash_recovery_area/ testdbds/*. Następnie, naciśnięto przycisk „*Finish*", co spowodowało rozpoczęcie procesu tworzenia bazy danych.

Tabela 5. Przebieg procedury tworzenia bazy danych.

3. Aplikacja testowa

3.1. Opis poglądowy

Aplikacja służy do zbierania, przetwarzania, ładowania oraz przechowywania informacji z gry *Ogame*, dostępnej przez przeglądarkę internetową. Jest ona zlokalizowana dla dużej liczby krajów. W obrębie jednej domeny (kraju) występuje wiele światów.

Na każdym z nich rywalizuje rzesza graczy, którzy mogą jednoczyć się w sojuszach. Polega ona na eksploracji przestrzeni kosmicznej, kolonizowaniu i rozbudowie planet oraz przeprowadzaniu ataków na innych graczy w celach zarobkowych, za pomocą zbudowanej flotylli. W grze dostępne są statystyki aktualizowane w czasie rzeczywistym. Dotyczą one zarówno sojuszy jak i kont graczy. Ponadto można wyróżnić stany punktowe różnego typu, które mają zastosowanie w kontekście sojuszu i konta gracza. Celem gry jest uzyskanie jak największej liczby punktów ogólnych. Sojusze oraz konta graczy zmieniają się w trakcie trwania rozgrywki. Gracz może zmienić nazwę lub przenieść swoją planetę macierzystą. Sojusz może zmienić nazwę lub znacznik.

Zadaniem aplikacji jest periodyczne zbieranie statystyk całego świata dla graczy i sojuszy, które egzystują na nim. Okres zbierania statystyk można ustawić na dowolny czas, co pozwala na skalowanie wielkości testowej bazy danych według uznania. W produkcyjnym zastosowaniu aplikacja będzie zabierać stany punktowe raz na dobę. Zakładając, że na jednym świecie rozgrywkę prowadzi 3000 graczy, którzy są zrzeszeni w 500 sojuszy, to w rezultacie pozwoli to jednorazowo na zgromadzenie 28 000 (8 stanów punktowych * 3000 graczy + 8 stanów punktowych * 500 sojuszy) stanów punktowych. W ciągu tygodnia pozwala to na załadowanie 196 000 wierszy. Ponadto, baza danych ma przechowywać informacje o wszystkich graczach i sojuszach wraz z pełną historią zmian, które zachodziły. Przy ładowaniu statystyk będą tworzone śladowe ilości wierszy historycznych. Pierwsze użycie programu zawsze prowadzi do utworzenia rekordu dla każdego sojuszu i konta.

Podsumowując, baza danych oraz oprogramowanie do przetwarzania i ładowania rekordów, muszą być dostosowane w pełni do modelowania realiów gry. Istotne, z punktu widzenia aplikacji, są następuje fakty:

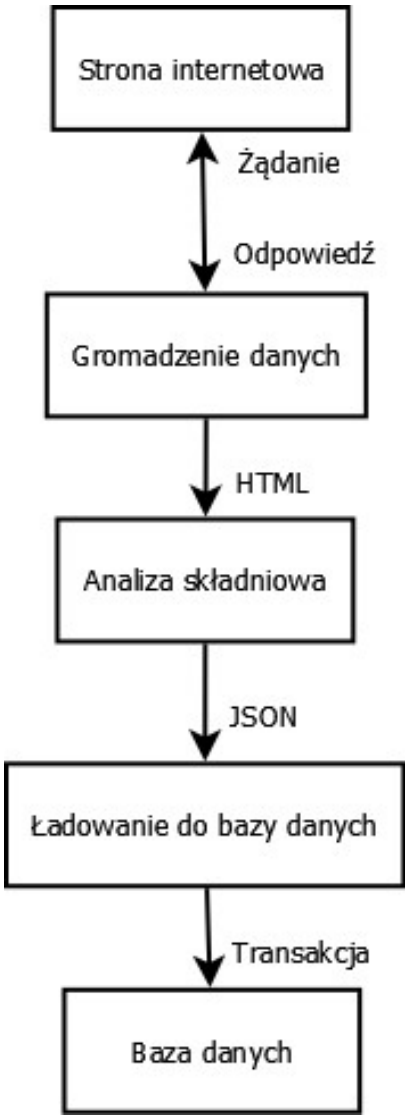
- Gra dostępna jest w wielu krajach.
- Gra toczy się na wielu niezależnych światach w obrębie jednego kraju.
- Dla każdego kraju nazwy światów są takie same.
- Każdy świat zawiera w sobie pewną ilość kont graczy i sojuszy.
- Każdy gracz może, ale nie musi należeć do sojuszu.
- Stany punktowe obejmują zarówno konta graczy jak i sojusze.
- Istnieje wiele typów stanów punktowych, które są stosowane w kontekście sojuszu i konta gracza.
- Statystyki są zbierane periodycznie.
- Dane kont graczy oraz sojuszy mogą zmieniać się.
- Encje posiadają szereg atrybutów niewymienionych w opisie, które należy przechowywać w bazie danych;

3.2. Masowe przetwarzanie i ładowanie

Aplikacja do masowego przetwarzania i ładowania stanów punktowych została zaimplementowana w języku *JAVA SE* i ma charakter wsadowy. Wybrane podejście zapewnia bezpieczeństwo, które należy rozumieć, jako uniknięcie możliwości utraty danych ze względu na skomplikowane procesy przetwarzania. Każdy etap jest realizowany w pełni niezależnie. Wyróżniono następujące etapy:

- Gromadzenie danych,
- Analiza składniowa,
- Ładowanie do bazy danych;

Na rysunku 1 został przedstawiony poglądowy schemat przepływu danych. Pokazuje on jak zmienia się postać informacji w trakcie procesu masowego przetwarzania i ładowania.



Rysunek 1. Przepływ informacji pomiędzy modułami aplikacji.

3.2.1. Gromadzenie danych

Podczas etapu gromadzenia statystyk, aplikacja generuje szereg żądań *HTTP* za pomocą programowej przeglądarki internetowej, w celu zalogowania się na specjalnie utworzone konto gracza oraz symulowania zachowania realnego użytkownika. Każda strona statystyk zawiera maksymalnie 100 kolejnych stanów punktowych. Stany punktowe są gromadzone na dwóch poziomach – konta gracza i sojuszu. Dla obu poziomów istnieje 8 typów stanu punktowego. Z powyższego wynika, że jednorazowe gromadzenie danych dla pojedynczego świata, powoduje wygenerowanie *ceil*(ilość graczy/100) * 8 + *ceil*(ilość sojuszy/100) * 8 żądań, gdzie funkcja *ceil* oznacza sufit. Wynikiem tego etapu są pliki *HTML* zawierające rezultaty, każdego wygenerowanego

żądania *HTTP*. Są one organizowane w specjalną hierarchię, aby zachować porządek. Lokalizacja zawiera informacje o dacie, obszarze gry, świecie i poziomie stanów punktowych.

```
./YYMMDD.HH24/IST/GAMEARENA/UNIVERSE/LEVEL/
```

Poniższa, przykładowa lokalizacja oznacza, że plik pochodzi z północy dnia 1.01.2013. Został on zarejestrowany dla świata nr 1 w Polsce i przechowuje stany punktowe na poziomie konta.

```
./130101.00/IST/POLAND/1.UNIVERSE/ACCOUNT
```

Nazewnictwo plików opiera się na poniższym formacie. Zawiera ono informacje o typie stanów punktowych, numerze strony oraz całkowitej ilości zarejestrowanych stron. Katalog nadrzędny determinuje to czy plik przechowuje stany punktowe graczy czy też sojuszy.

```
TYPE.NNN.NNN.IST
```

Przykładowa nazwa pliku mówi o nim, że przechowuje ekonomiczne stany punktowe, dla graczy lub sojuszy, którzy zajmowali pozycje od 401-500. Łącznie na świecie istnieje od 2701 do 2800 graczy lub sojuszy.

```
ECONOMY.005.028.IST
```

Statystyki są aktualizowane w czasie rzeczywistym. Aby zapobiec potencjalnym problemom należy jak najszybciej pobrać obraz statystyk jednego typu. Oznacza to, że w jak najkrótszym czasie należy wygenerować *ceil(*ilość graczy/100*)* żądań, zbuforować wyniki i dopiero je przetworzyć. Tak też czyni aplikacja. Nie gwarantuje to jednak braku możliwości wystąpienia dwóch błędnych scenariuszy. W pierwszym z nich, jeden gracz może pojawić się podwójnie w jednym zestawieniu:

1. Gracz A znajduje się na pozycji 499.
2. Program zrzuca obraz 5 strony statystyk, która przechowuje graczy z pozycji 401-500.
3. Gracz A spada na pozycję 501.
4. Program zrzuca obraz 6 strony statystyk, która przechowuje graczy z pozycji 501-600.

Analogicznym przypadkiem jest sytuacja, w której gracz w ogóle nie pojawi się w statystykach:

1. Gracz A znajduje się na pozycji 401.
2. Program zrzuca obraz 4 strony statystyk, która przechowuje graczy z pozycji 301-400.
3. Gracz A awansuje na pozycję 399.
4. Program zrzuca obraz 5 strony statystyk, która przechowuje graczy z pozycji 401-500.

Szansa na wystąpienie w/w scenariuszy jest znikoma, ale możliwa. Z tego też względu należy gromadzić statystyki częściej niż raz na dobę, aby dysponować rezerwowym obrazem. W praktyce, obrazy są wykonywane codziennie w godzinach 18-6, co godzinę. Jeśli obraz z północy okaże się błędny, to należy załadować inny, najbliższy tj. z godziny 23 lub 1. Analogicznie należy postąpić, gdy obrazy z godziny 23 i 1 też są uszkodzone. Gromadzenie 12 obrazów na dobę, daje niemal 100% pewności, że któryś z nich jest prawidłowy. W trakcie testów ani razu nie udało się zaobserwować potencjalnych błędów.

3.2.2. Analiza składniowa

W trakcie analizy składniowej sprawdzany jest format plików *HTML* stworzonych przez pierwszy moduł, który odpowiada za gromadzenie statystyk. Jeśli analizator składniowy potrafi przetworzyć informacje, zrobi to. W przeciwnym przypadku zostanie wygenerowany błąd, którego obsługa leży już w rękach twórcy oprogramowania. Niekiedy mogą zajść zmiany w grze, które sprawią, że wyodrębnienie istotnych informacji ze źródła strony internetowej nie będzie możliwe. Zachowanie oryginalnych plików z zawartością żądań, pozwala na dostosowanie analizatora do nowego formatu wejściowego i ponowne przetworzenie. Dla każdego pliku zawierającego surowe dane w *HTML*, tworzony jest plik przechowujący te same rekordy i dane, ale w postaci *JSON*. Hierarchia i nazewnictwo plików są identyczne jak w przypadku gromadzenia danych. Jedyną różnicą jest to, że pliki wyniki parsowania są przechowywane w gałęzi *PST* zamiast *IST*. Analogicznie, rozszerzenie plików to *PST* a nie *IST*. Poniżej znajduje się przykładowy rekord na poziomie konta gracza, w postaci JSON.

```
{
   "allianceExternalId": 40153,
   "allianceTag": "P N",
```

```
   "checkpoint": "130101.00",
   "externalId": 402886,
   "gameArea": "POLAND",
   "homePlanetGalaxy": 9,
   "homePlanetPosition": 9,
   "homePlanetSolarSystem": 470,
   "honorPoints": 774421,
   "level": "ACCOUNT",
   "name": "HarryBrown",
   "points": 774421,
   "position": 1,
   "positionChange": 0,
   "type": "HONOR",
   "universeName": "1.UNIVERSE"
}
```

Rekord dla sojuszu jest uboższy i zawiera mniej informacji. Poniżej znajduje się przykładowy rekord na poziomie sojuszu, w postaci *JSON*.

```
{
   "averagePoints": 345523,
   "checkpoint": "130101.00",
   "externalId": 41586,
   "gameArea": "POLAND",
   "level": "ALLIANCE",
   "name": "Bandits",
   "points": 345523,
   "position": 301,
   "positionChange": 0,
   "numberOfMebers": 1,
   "tag": "Bandits",
   "type": "ECONOMY",
   "universeName": "1.UNIVERSE"
}
```

3.2.3. Ładowanie do bazy danych

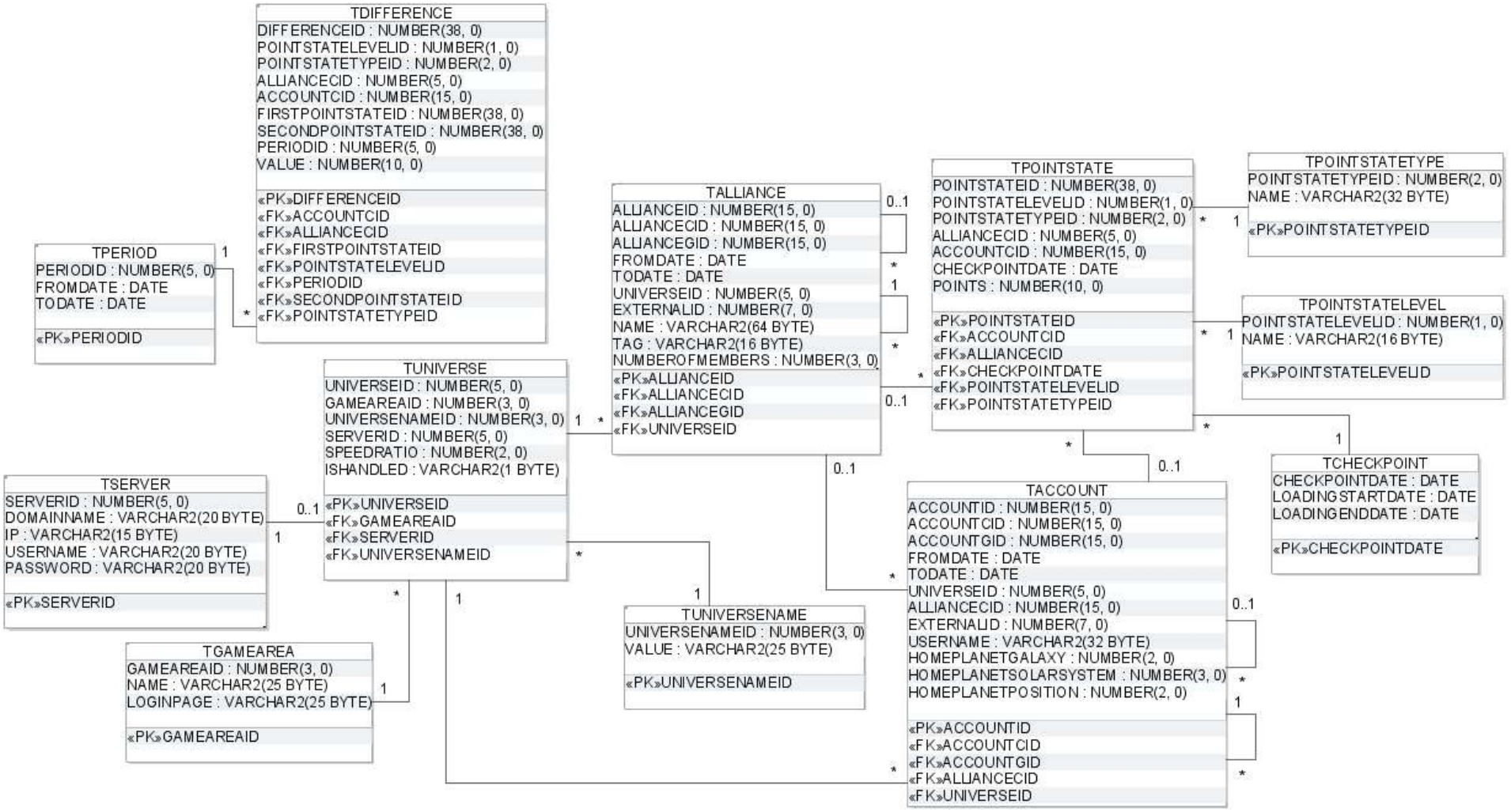
W trakcie ładowania do bazy danych, poszczególne rekordy są wstawiane do tabel za pośrednictwem interfejsu, stworzonego w *PL/SQL*. Logika encji, odpowiedzialna za kontrolę integralności oraz historii, umieszczana jest w pakietach, o tej samej nazwie, co encja, ale z prefiksem „P”. Takie podejście pozwala na szybką implementację oraz skorzystanie w przyszłości z obiektów oraz widoków obiektowych, poprzez import istniejącej logiki z ciała pakietu do metod obiektu. Atomowa operacja wstawienia stanu punktowego polega na weryfikacji czy dany gracz lub sojusz istnieje. Jeśli istnieje to należy sprawdzić czy zaszły jakiekolwiek zmiany oraz zaktualizować rejestr historii, jeśli jest to konieczne. W przeciwnym przypadku należy stworzyć nowy rekord. Gracze i sojusze są odszukiwani za pomocą zewnętrznego identyfikatora, który jest wyłuskiwany na etapie analizy składniowej. Nie ulega on zmianie przez cały czas istnienia danego

konta lub sojuszu. Po załadowaniu wszystkich stanów punktowych unieważniane są konta
i sojusze, dla których nie pojawił się żaden stan punktowy w aktualnie przetwarzanym znaczniku czasowym. Oznacza to, że nie istnieją już one. Następnie weryfikowana jest integralność bazy danych poprzez sprawdzenie czy rozkład stanów punktowych ze względu na typ jest dokładnie równomierny. Jeśli ilości rekordów dla poszczególnych typów stanu punktowego są różne, transakcja zostaje wycofana i podnoszony jest wyjątek. Weryfikacja pozwala na wykrycie błędnych scenariuszy opisanych w podrozdziale 3.2.1.

3.3. Schemat bazy danych

Po przeprowadzeniu gruntownej analizy wymagań oraz gry, zaprojektowano schemat bazy danych przedstawiony na rysunku 2. Dla tabeli *TDIFFERENCE* nie pokazano większości zależności, ponieważ służy ona do generowania raportów i jest zdenormalizowana. Wszystkie obiekty związane z aplikacją testową zostały umieszczone w schemacie *OIMS*. W trakcie ustalania nazw przyjęto następujące konwencje nazewnicze:

- Wszystkie nazwy obiektów i atrybutów są w języku angielskim.
- Poszczególne słowa w nazwach obiektów i atrybutów tabel nie są oddzielone separatorem.
- Nazwy tabel zaczynają się od prefiksu „T” a potem następuje właściwa nazwa encji w liczbie pojedynczej.
- Nazwy kluczy głównych składają się z nazwy encji w liczbie pojedynczej i postfiksu „ID”.
- Atrybuty służące do budowania historii w tabeli składają się z nazwy encji w liczbie pojedynczej i postfiksu „CID” lub „GID”.
- Klucze obce nazywają się tak samo jak pola kluczy głównych w tabelach, do których odwołują się one.
- Jeśli klucz obcy odnoszący się do tej samej tabeli jest stosowany wielokrotnie w obrębie jednej tabeli to wtedy ich nazwy są poprzedzone prefiksem odróżniającym np. *FIRSTCHECKPOINTDATE* i *SECONDCHECKPOINTDATE*.



Rysunek 2. Schemat testowej bazy danych

Baza danych ma aktualnie charakter hurtowni danych. Nie występują w niej współbieżne transakcje a jej główne zastosowanie polega na odpytywaniu tabel, w celu uzyskania zestawień statystycznych graczy i sojuszy. Dane są ładowane masowo raz na dobę. Strukturę bazy danych można podzielić na cztery logiczne części:

- Stałą,
- Ciągle rozbudowywaną i aktualizowaną,
- Ciągle rozbudowywaną,
- Raportową;

Podział ten należy wziąć pod uwagę podczas późniejszego dostrajania bazy danych. Dla każdej części powinny zostać dobrane odpowiednie parametry przestrzeni tabel. W skład części stałej wchodzą relacje wymienione w tabeli 6. Obiekty *TGAMEAREA*, *TUNIVERSENAME*, *TSERVER* są wykorzystywane, jako źródło konfiguracji dla oprogramowania do masowego przetwarzania i ładowania. Tabela *TUNIVERSE*, za pośrednictwem swojego klucza głównego, jest łącznikiem pomiędzy częścią stałą a resztą bazy danych. Tabele *TPOINTSTATELEVEL* i *TPOINTSTATETYPE* powstały w procesie normalizacji. We wszystkich tabelach, należących do tejże grupy, zmiany zachodzą bardzo rzadko.

Nazwa	**Zawartość**
TGAMEAREA	Obszary rozgrywki - mogą to być kraje, kontynenty i areny międzynarodowe.
TUNIVERSENAME	Nazwy światów - pula jest wspólna dla wielu obszarów rozgrywki.
TSERVER	Dane serwerów, wraz z nazwą gracza i hasłem dla konta, które jest wykorzystywane do gromadzenia stanów punktowych.
TUNIVERSE	Obsługiwane światy – flaga *ISHANDLED*, decyduje czy oprogramowanie ma gromadzić stany punktowe.
TPOINTSTATELEVEL	Poziomy stanu punktowego – obecnie istnieją stany punktowe na poziomie konta i sojuszu.
TPOINTSTATETYPE	Typy stanów punktowych – obecnie istnieje 8 typów stanów punktowych. Są to punkty ogólne, ekonomiczne, badań, militarne aktualnie zbudowane, militarne zbudowane przez całą grę, militarne stracone przez całą grę, militarne zniszczone przez całą grę oraz honoru.

Tabela 6. Tabele rzadko rozbudowywane i aktualizowane.

Część ciągle rozbudowywana i aktualizowana wzrasta przy każdym masowym ładowaniu stanów punktowych. Relacje należące do tej grupy są przedstawione w tabeli

7. W ich przypadku, naturalną operacją są też aktualizacje. Parametry inicjalizacyjne przestrzeni tabel powinny brać to pod uwagę, aby nie dopuścić do łańcuchowania i migracji wierszy.

Nazwa	**Zawartość**
TALLIANCE	Sojusze wraz z pełną historią.
TACCOUNT	Konta graczy wraz z pełną historią.

Tabela 7. Tabele ciągle rozbudowywane i aktualizowane.

Część ciągle rozbudowywana wzrasta przy każdym masowym ładowaniu stanów punktowych. Relacje należące do tej grupy są przedstawione w tabeli 8. Ich wiersze nigdy nie są modyfikowane. Naturalnymi operacjami są tylko instrukcje *INSERT* i *DELETE*. Wiedza ta pozwala na efektywniejsze przechowywanie wierszy w blokach, ponieważ nie jest wymagane zarezerwowane wolne miejsce na potrzeby przyszłych aktualizacji. Pełny odczyt tabeli *TPOINTSTATE* będzie mniej kosztowny, jeśli wiersze będą gęsto upakowane w blokach.

Nazwa	**Zawartość**
TCHECKPOINT	Znaczniki czasowe załadowanych obrazów stanów punktowych.
TPOINTSTATE	Stany punktowe – dla wszystkich poziomów i typów.
TPERIOD	Charakterystyczne okresy wraz z datą początkową i końcową – tygodnie, miesiące, lata.

Tabela 8. Tabele ciągle rozbudowywane.

Część raportowa wzrasta tylko wtedy, gdy należy wyznaczyć statystyki dla okresu zdefiniowanego w tabeli *TPERIOD*. Aktualnie, w skład tej grupy wchodzi tylko jedna relacja przedstawiona w tabeli 9. Obiekt *TDIFFERENCE* został zdenormalizowany w celu zwiększenia wydajności. Zawiera on dane w prawie takiej samej postaci, jaka jest żądana przez użytkownika.

Nazwa	**Zawartość**
TDIFFERENCE	Różnice pomiędzy dwoma stanami punktowymi tego samego poziomu i typu, dla tego samego gracza lub sojuszu. Jeśli stan punktowy z jego znacznika czasowego nie posiada swojego odpowiednika w drugim znaczniku czasowym to nie da się wyznaczyć różnicy i rekord nie powstaje.

Tabela 9. Tabele raportowe.

3.4. Przygotowanie zbioru danych

Aby uzyskać zbiór danych, wykonano masowe ładowanie stanów punktowych z dni 17-24.12.2012. Ładowanie wszystkich ośmiu obrazów statystyk, z jednego świata gry, zakończyło się sukcesem. Liczności wierszy w poszczególnych tabelach schematu *OIMS* bazy danych zostały przedstawione w tabeli 10.

Nazwa tabeli	**Ilość rekordów**
TCHECKPOINT	8
TALLIANCE	492
TACCOUNT	2869
TPOINTSTATE	204912
TDIFFERENCE	25064

Tabela 10. Ilości rekordów w tabelach po załadowaniu danych.

4. Nowe funkcjonalności

4.1. Wirtualne kolumny

Wartość kolumny wirtualnej jest obliczana w locie na podstawie wyrażenia, które może zawierać wywołania funkcji. Dzięki temu rezultat nie jest trwale przechowywany i nie zajmuje przestrzeni dyskowej. Wyrażenie determinujące wartość należy podać w trakcie definiowania kolumny wirtualnej. Dla wyrażenia wprowadzono następujące ograniczenia [10]:

- Jeśli następuje wywołanie funkcji użytkownika to musi być ona jawnie zadeklarowana, jako *DETERMINISTIC*. Restrykcja oznacza, że dla dokładnie tych samych argumentów zwraca ona za każdym razem dokładnie ten sam wynik.
- Rezultat musi być w każdym przypadku typem skalarnym. Nie może to być typ złożony, zdefiniowany przez użytkownika lub wielki obiekt.
- Nie może odnosić się do innej kolumny wirtualnej, nawet, jeśli znajduje się ona w tej samej tabeli.
- Nie może odnosić się do rzeczywistych kolumn spoza macierzystej tabeli.

Kolumnę wirtualną można zdefiniować przy tworzeniu nowej tabeli lub podczas edycji istniejącej (*DDL*). Tylko relacyjne tabele mogą korzystać z wirtualnych kolumn. Inne dostępne rodzaje tabel tj. indeksowe, zewnętrzne, obiektowe, klastrowe i tymczasowe, nie są zdolne do obsługi nowej funkcjonalności. Wirtualna kolumna może pojawić się w zapytaniach, jako warunek filtrujący, a także przy zmianie rekordów, jako warunek modyfikacji lub usunięcia wierszy (*DML*). Nie można ustawić wartości kolumny wirtualnej poprzez operację *UPDATE*.

Pomimo, że funkcja, wykorzystywana przez wyrażenie kolumny wirtualnej, musi być jawnie zadeklarowana, jako deterministyczna, istnieje możliwość obejścia tej restrykcji [10]. Ograniczenie jest weryfikowane tylko w trakcie tworzenia wirtualnej kolumny. Po jej zdefiniowaniu można zmodyfikować funkcję tak, aby nie była jawnie zadeklarowana jako deterministyczna. Nie spowoduje to unieważnienia kolumny wirtualnej. Opisana recepta sprawia jednak, że kolumna wirtualna może zwracać

nieprawidłowe wartości. Aby mieć pewność, że rezultat zwracany przez kolumnę wirtualną jest wiarygodny należy wykonać następujące czynności:

- Wyłączenie i włączenie ograniczeń utworzonych dla kolumny wirtualnej.
- Przebudowanie indeksów utworzonych dla kolumny wirtualnej.
- Pełne odświeżenie widoków zmaterializowanych, które korzystają z kolumny wirtualnej.
- Przeładować bufor, który może przechowywać wartości kolumny wirtualnej.
- Ponownie zebrać statystyki dla tabeli zawierającej kolumnę wirtualną.

Aby uzyskać informację o tym czy dane konto zostało skasowane, należy zbadać wartość pola *ACCOUNTCID* oraz *TODATE* dla ostatniego wpisu z danej grupy. Jeśli w *ACCOUNTCID* znajduje się liczba większa od 0 oraz *TODATE* przechowuje datę z przeszłości, konto nie istnieje. Jego rekordy są przechowywane tylko w celach historycznych. W przyszłości, plan utrzymania tabel będzie definiował, co i po jakim czasie ma się dziać z tymi wpisami. Nie ma jednak w tabeli jawnej informacji o tym czy dany wpis należy do grupy usuniętych. Może ona zostać umieszczona w kolumnie wirtualnej. Jeśli zostanie wykorzystana funkcja niezadeklarowana jawnie, jako *DETERMINISTIC* to pojawia się błąd „*ORA-30553: The function is not deterministi*c". Za sprawdzenie czy dany wpis dotyczy nieistniejącego konta odpowiada poniższa funkcja.

```
CREATE OR REPLACE
  FUNCTION ISACCOUNTROWDELETED(
      PNACCOUNTGID NUMBER)
    RETURN VARCHAR2 DETERMINISTIC
  IS
    VNCOUNT NUMBER;
  BEGIN
    SELECT
      COUNT(*)
    INTO
      VNCOUNT
    FROM
      TACCOUNT
    WHERE
      ACCOUNTGID   = PNACCOUNTGID
    AND TODATE     < SYSDATE
    AND ACCOUNTCID = ACCOUNTGID;
    IF VNCOUNT     = 0 THEN
      RETURN('0');
    ELSE
      RETURN('1');
    END IF;
  END ISACCOUNTROWDELETED;
```

Tabela *TACCOUNT* została zmieniona poprzez dodanie kolumny wirtualnej. Wartość atrybutu *ISDELETED* będzie równa „1” dla wszystkich wierszy historycznych, które należą do nieistniejących już kont graczy.

```
ALTER TABLE TACCOUNT ADD (ISDELETED AS
(ISACCOUNTROWDELETED(ACCOUNTGID)));
```

Należy zwrócić uwagę na to, że istnienie kolumny wirtualnej powoduje pewne problemy przy instrukcji *INSERT*. Trzeba użyć słowa kluczowego *DEFAULT* w miejscu gdzie definiuje się wartość kolumny wirtualnej. Inną możliwością jest zastosowanie pełnej formy polecenia, w której wyliczone są wszystkie nazwy zwykłych kolumn. Wykorzystana funkcja nie jest deterministyczna z matematycznego punktu widzenia, ponieważ wartość dla tego samego argumentu może zmieniać się w czasie. Należy usunąć słowo kluczowe *DETERMINISTIC* z definicji. Nie spowoduje to unieważnienia kolumny wirtualnej opartej o tę funkcję. Próba zmiany wartości wirtualnej za pomocą instrukcji *UPDATE* skutkuje wystąpieniem błędu „*ORA-54017: UPDATE operation disallowed on virtual columns*”. Poniższe zapytanie zwraca wszystkie rekordy, które są związane z nieistniejącymi kontami.

```
 SELECT * FROM TACCOUNT WHERE ISDELETED  = '1'
```

Aby przetestować czy wartości kolumny wirtualnej zmieniają się prawidłowo wykonano test. Zaktualizowano konto jednego gracza tak, aby kwalifikowało się, jako usunięte. Po aktualizacji sprawdzono wartość kolumny *ISDELETED*. Zmieniła się ona z „0” na „1”, co świadczy o tym, że aktualizacja wartości kolumny wirtualnej nastąpiła prawidłowo a SZBD nie odczytał starej wartości przechowywanej w buforze.

```
UPDATE TACCOUNT SET TODATE = TO_DATE('19-12-12') WHERE USERNAME =
'Destro';
```

Następnie zaktualizowano konto tak, aby jego stan był taki sam jak na początku. Wartość kolumny wirtualnej również zmieniła się z „1” na „0”.

```
UPDATE TACCOUNT SET TODATE = NULL WHERE USERNAME = 'Destro'
```

Gdyby na tym kończyły się możliwości wirtualnych kolumn, byłyby one niewielkim udogodnieniem w stosunku do widoków, które pozwalają na rozszerzenie wyniku, pobranego ze zwykłej tabeli, o dodatkowe kolumny obliczane na podstawie wyrażeń. Kolumny wirtualne należy traktować jak zwykłe. Mogą one być indeksowane, analizowane w celu gromadzenia statystyk, weryfikowane przez ograniczenia oraz wykorzystywane, jako klucze partycjonowania. Od wersji *Oracle Database 11g Release*

2 możliwe jest utworzenie ograniczeń klucza głównego i obcego [6]. Kolumny wirtualne korzystają z indeksów funkcyjnych. Zasada działania tychże indeksów jest opisana literaturze [11]. Aby partycjonowanie z wykorzystaniem kolumny wirtualnej, jako klucza było możliwe, w wyrażeniu nie mogą pojawić się funkcje *PL/SQL*, w tym również te zdefiniowane przez użytkownika. Jest to niewątpliwie spora niedogodność.

4.2. Operatory transformacji

Wprowadzono operatory *PIVOT* i *UNPIVOT*, które są rozszerzeniem klauzuli *FROM* instrukcji *SELECT* [10]. Operacja *PIVOT* grupuje wiele rekordów w jeden wiersz, zamieniając wiersze na kolumny. Relacyjny rezultat zapytania transformowany jest do postaci czytelnej dla zwykłego człowieka. Formatuje wynik tak, aby zawierał mniej wierszy kosztem większej ilości kolumn, jednocześnie wprowadzając logiczną agregację skojarzonych ze sobą danych. Ponadto operatory upraszczają zapytania w znaczący sposób. Przed ich wprowadzeniem, ten sam rezultat można było osiągnąć jedynie poprzez wprowadzanie wielu zapytań zagnieżdżonych w klauzuli *SELECT*, dla każdej dodatkowej kolumny, która miała się pojawić.

Operator *UNPIVOT* działa dokładnie odwrotnie do *PIVOT*. Pozwala dokonać transformacji formatu czytelnego dla człowieka do postaci relacyjnej. Uproszcza procedurę wprowadzania do bazy danych informacji dostarczonych przez dział administracji. Po konwersji nie pozostaje nic innego, jak dokonać wstawienia rekordów do tabeli.

4.2.1. Operator *PIVOT*

Aby zaprezentować działanie operatora *PIVOT*, został wygenerowany raport dziennych stanów punktowych w ciągu tygodnia. Wynik ograniczono do dwóch graczy. Dla jednego z nich, celowo nie są wybrane stany punktowe z każdego dnia. Rezultat zapytania został umieszczony w tabeli 11.

```
SELECT
  AC.USERNAME,
  PS.CHECKPOINTDATE,
  PS.POINTS
FROM
  TPOINTSTATE PS
JOIN TALLIANCE AL
ON
  PS.ALLIANCECID = AL.ALLIANCECID
JOIN TACCOUNT AC
ON
  PS.ACCOUNTCID = AC.ACCOUNTCID
WHERE
    AL.TAG               = 'A.PMG'
AND PS.POINTSTATETYPEID = 1
AND AC.USERNAME        IN ('Destro', 'rudven')
ORDER BY
  AC.USERNAME;
```

POINTS	USERNAME	CHECKPOINTDATE
7474582	Destro	17-12-12 00:00:00
7476256	Destro	18-12-12 00:00:00
7476256	Destro	19-12-12 00:00:00
7553288	Destro	20-12-12 00:00:00
7553288	Destro	21-12-12 00:00:00
7553288	Destro	22-12-12 00:00:00
7578854	Destro	23-12-12 00:00:00
14346	Rudven	17-12-12 00:00:00
14346	Rudven	18-12-12 00:00:00

Tabela 11. Stany punktowe przed zastosowaniem operacji PIVOT.

W tabeli 12 można zaobserwować rezultat po transformacji operatorem PIVOT względem kolumny „*Data*” dla wartości „*Stan punktowy*”. Wybór kolumny względem, której nastąpi operacja jest w rękach programisty. Powinien on zdawać sobie sprawę z tego czy operacja ma sens oraz czy wpłynie pozytywnie na prezentację raportu. W przykładzie, udało się uzyskać zdecydowanie bardziej przyjazny wynik. Trzy kolumny i dziewięć wierszy zastąpiono ośmioma kolumnami i dwoma wierszami. Należy zwrócić uwagę, że kolumna względem, której następuje obrót, musi zostać poddana działaniu funkcji agregującej, tak, aby wynikiem była zawsze jedna wartość. Istnieje możliwość, że dla tego samego gracza i tego samego znacznika czasowego wystąpi wiele stanów punktowych. W przykładzie takiej sytuacji nie ma, ponieważ warunek ograniczający (*ps.pointstatetypeid = 1*) nie pozwala na to. Jeśli funkcja agregująca nie zostanie użyta to wystąpi błąd „*ORA-56902: expect aggregate function inside pivot operation*”. Aby temu zapobiec, skorzystano z neutralnej funkcji *MAX*, która dla zbioru jednoelementowego

zawsze zwróci dokładnie ten jeden element. Niestety w klauzuli *IN* nie można wykorzystać podzapytania. Wartości muszą zostać jawnie zdefiniowane, co jest sporym utrudnieniem w utrzymaniu zapytania. Nazwy kolumn w klauzulach *PIVOT* i *FOR* nie mogą być poprzedzone nazwą obiektu, z którego pochodzą. Jeśli ta reguła nie jest zachowana to występuję błąd „*ORA-01748: only simple column names allowed here*". W przypadku złożonych zapytań należy zagnieździć zapytanie tak, aby operator *PIVOT* widział rezultat, jako prostą tabelę.

```
SELECT
  *
FROM
  (
    SELECT
      AC.USERNAME,
      PS.CHECKPOINTDATE,
      PS.POINTS
    FROM
      TPOINTSTATE PS
    JOIN TALLIANCE AL
    ON
      PS.ALLIANCECID = AL.ALLIANCECID
    JOIN TACCOUNT AC
    ON
      PS.ACCOUNTCID = AC.ACCOUNTCID
    WHERE AL.TAG              = 'A.PMG'
    AND   PS.POINTSTATETYPEID = 1
    AND   AC.USERNAME        IN ('Destro', 'rudven')
    ORDER BY
      AC.USERNAME;
  )
  PIVOT (MAX(POINTS)
AS
  PTS FOR (CHECKPOINTDATE) IN ( TO_DATE('17-12-12', 'DD-MM-YY')
AS
  MON, TO_DATE('18-12-12', 'DD-MM-YY')
AS
  TUE, TO_DATE('19-12-12', 'DD-MM-YY')
AS
  WED, TO_DATE('20-12-12', 'DD-MM-YY')
AS
  THU, TO_DATE('21-12-12', 'DD-MM-YY')
AS
  FRI, TO_DATE('22-12-12', 'DD-MM-YY')
AS
  SAT, TO_DATE('23-12-12', 'DD-MM-YY')
AS
  SUN)) ORDER BY USERNAME;
```

USERNAME	MON_PTS	TUE_PTS	WED_PTS	THU_PTS	FRI_PTS	SAT_PTS	SUN_PTS
Destro	7474582	7476256	7476256	7553288	7553288	7553288	7578854
rudven	14346	14346					

Tabela 12. Stany punktowe po zastosowaniu operacji PIVOT.

Rezultat z tabeli 12 można osiągnąć za pomocą konwencjonalnego zapytania, które jest trochę bardziej rozbudowane, mniej czytelne oraz trudniejsze w utrzymaniu.

```
SELECT
  AC.USERNAME,
  MON_PTS,
  TUE_PTS,
  WED_PTS,
  THU_PTS,
  FRI_PTS,
  SAT_PTS,
  SUN_PTS
FROM
  (
    SELECT
      ACCOUNTCID ACID,
      ALLIANCECID ALCID,
      MAX(DECODE (CHECKPOINTDATE, TO_DATE('17-12-12', 'DD-MM-YY'),
POINTS, NULL)) MON_PTS,
      MAX(DECODE (CHECKPOINTDATE, TO_DATE('18-12-12', 'DD-MM-YY'),
POINTS, NULL)) TUE_PTS,
      MAX(DECODE (CHECKPOINTDATE, TO_DATE('19-12-12', 'DD-MM-YY'),
POINTS, NULL)) WED_PTS,
      MAX(DECODE (CHECKPOINTDATE, TO_DATE('20-12-12', 'DD-MM-YY'),
POINTS, NULL)) THU_PTS,
      MAX(DECODE (CHECKPOINTDATE, TO_DATE('21-12-12', 'DD-MM-YY'),
POINTS, NULL)) FRI_PTS,
      MAX(DECODE (CHECKPOINTDATE, TO_DATE('22-12-12', 'DD-MM-YY'),
POINTS, NULL)) SAT_PTS,
      MAX(DECODE (CHECKPOINTDATE, TO_DATE('23-12-12', 'DD-MM-YY'),
POINTS, NULL)) SUN_PTS
    FROM
      TPOINTSTATE
    GROUP BY
      ACCOUNTCID,
      ALLIANCECID
  )
JOIN TACCOUNT AC
ON
  AC.ACCOUNTCID = ACID
JOIN TALLIANCE ALd
ON
  AL.ALLIANCECID = ALCID
WHERE
  AL.TAG         = 'A.PMG'
AND AC.USERNAME IN ('Destro', 'rudven');
```

Testy wydajności wykonano tak, aby wyeliminować wpływ buforowania na wynik. Oba zapytania zostały wykonane na świeżo uruchomionej instancji bazy danych. Wykorzystanie operatora *PIVOT* jest widoczne w planie wykonania zapytania, jako operacja *GROUP BY PIVOT*, co może świadczyć o zoptymalizowanej ścieżce dostępu do danych. Czasy wykonania znajdują się w tabeli 13.

Zapytanie	Czas pierwszego wykonania [s]	Czas kolejnych wykonań [s]
Z operatorem *PIVOT*	0,063	0,009
Bez operatora *PIVOT*	0,194	0,082

Tabela 13. Porównanie wydajności dwóch zapytań, bez i z operatorem *PIVOT*.

4.2.1. Operator *PIVOT* z opcją *XML*

Zapytanie, korzystające z operatora *PIVOT*, rozbudowano o opcję *XML*. Sprawia ona, że kolumny, na które wpływa operator, zostaną zastąpione jedną kolumną typu *XMLTYPE*. Wynik zostaje skonwertowany do formatu *XML* i przedstawiony, jako wartość tejże kolumny. Przykładowy rezultat został umieszczony w tabeli 14. Ponadto, zapytanie staje się bardziej elastyczne, ponieważ w klauzuli *IN* może pojawić się podzapytanie lub słowo kluczowe *ANY*, które dopuszcza każdą wartość występującą w kolumnie, względem, której następuje obrót.

```
SELECT
  *
FROM
  (
    SELECT
      AC.USERNAME,
      PS.CHECKPOINTDATE,
      PS.POINTS
    FROM
      TPOINTSTATE PS
    JOIN TALLIANCE AL
    ON
      PS.ALLIANCECID = AL.ALLIANCECID
    JOIN TACCOUNT AC
    ON
      PS.ACCOUNTCID = AC.ACCOUNTCID
    WHERE
      AL.TAG                 = 'A.PMG'
    AND PS.POINTSTATETYPEID = 1
    AND AC.USERNAME        IN ('Destro', 'rudven')
    ORDER BY
      AC.USERNAME
  )
  PIVOT XML (MAX(POINTS) AS PTS FOR (CHECKPOINTDATE) IN (ANY))
ORDER BY
  USERNAME DESC;
```

USERNAME	CHECKPOINTDATE_XML
rudven	<PivotSet><item><column name = "CHECKPOINTDATE">2012-12-17</column><column name = "PTS">14346</column></item><item><column name = "CHECKPOINTDATE">2012-12-18</column><column name = "PTS">14346</column></item></PivotSet>

USERNAME	CHECKPOINTDATE_XML
Destro	<PivotSet><item><column name = "CHECKPOINTDATE">2012-12-17</column><column name = "PTS">7474582</column></item><item><column name = "CHECKPOINTDATE">2012-12-18</column><column name = "PTS">7476256</column></item><item><column name = "CHECKPOINTDATE">2012-12-19</column><column name = "PTS">7476256</column></item><item><column name = "CHECKPOINTDATE">2012-12-20</column><column name = "PTS">7553288</column></item><item><column name = "CHECKPOINTDATE">2012-12-21</column><column name = "PTS">7553288</column></item><item><column name = "CHECKPOINTDATE">2012-12-22</column><column name = "PTS">7553288</column></item><item><column name = "CHECKPOINTDATE">2012-12-23</column><column name = "PTS">7578854</column></item><item><column name = "CHECKPOINTDATE">2012-12-24</column><column name = "PTS">7578942</column></item></PivotSet>

Tabela 14. Rezultat operatora *PIVOT* z opcją *XML*.

4.2.2. Operator *UNPIVOT*

Rezultat uzyskany poprzez operator *PIVOT* zapisano w tabeli *TPOINTSTATEREPORT*. Posłuży ona do wykonania odwrotnej transformacji z wykorzystaniem operatora *UNPIVOT*. Poniższe zapytanie zwróciło dokładnie taki sam wynik jak zawartość tabeli 11, ponieważ domyślnie w operacji *UNPIVOT* wykorzystana jest opcja *EXCLUDE NULLS*. W klauzuli *IN* definiuje się kolumny, które powinny zostać przetransformowane na wiersze. Zdefiniowanie aliasu jest obowiązkowe, ponieważ wtedy mechanizm wie, jakie wartości należy przypisać polu *CHECKPOINTDATE* dla poszczególnych stanów punktowych.

```
SELECT
  *
FROM
  TPOINTSTATEREPORT UNPIVOT (POINTS FOR CHECKPOINTDATE IN (
MON_PTS AS '17-12-12',
TUE_PTS AS '18-12-12',
WED_PTS AS '19-12-12',
THU_PTS AS '20-12-12',
FRI_PTS AS '21-12-12',
SAT_PTS AS '22-12-12',
SUN_PTS AS '23-12-12'));
```

Aby uwidocznić wiersze, których wartość punktowa jest równa *NULL* skorzystano z opcji *INCLUDE NULLS*. Rezultat umieszczono w tabeli 15. Zgodnie z oczekiwaniami, wiersze o wartości *NULL*, zostały dołączone do wyniku.

```
SELECT
  *
FROM
  TPOINTSTATEREPORT UNPIVOT INCLUDE NULLS (POINTS FOR CHECKPOINTDATE
IN (
MON_PTS AS '17-12-12',
TUE_PTS AS '18-12-12',
WED_PTS AS '19-12-12',
THU_PTS AS '20-12-12',
FRI_PTS AS '21-12-12',
SAT_PTS AS '22-12-12',
SUN_PTS AS '23-12-12'));
```

USERNAME	CHECKPOINTDATE	POINTS
Destro	17-12-12 00:00:02	7474582
Destro	18-12-12 00:00:01	7476256
Destro	19-12-12 00:00:01	7476256
Destro	19-12-12 00:00:02	7553288
Destro	21-12-12 00:00:01	7553288
Destro	22-12-12 00:00:01	7553288
Destro	23-12-12 00:00:02	7578854
rudven	17-12-12 00:00:02	14346
rudven	18-12-12 00:00:01	14346
rudven	19-12-12 00:00:01	
rudven	19-12-12 00:00:02	
rudven	21-12-12 00:00:01	
rudven	22-12-12 00:00:01	
rudven	23-12-12 00:00:02	

Tabela 15. Stany punktowe po zastosowaniu *UNPIVOT* z opcją *INCLUDE NULLS*.

Aby uzyskać podobny rezultat bez wykorzystania operatora *UNPIVOT*, doprowadzono do powstania iloczynu kartezjańskiego tak, aby uzyskać żądaną liczbę wierszy poprzez bezwarunkowe złączenie. Instrukcja *DECODE* powoduje, że wyświetla się prawidłowy stan punktowy w każdym wierszu. Wykluczenie wierszy z wartością równą *NULL* jest w tym przypadku wykonywalne za pomocą zwykłego warunku w klauzuli *WHERE*.

```
SELECT
  PSR.USERNAME,
  RD.REGISTEREDDATE AS CHECKPOINTDATE,
  DECODE(CHL.LOADEDCHECKPOINTDATE,
  TO_DATE('17-12-12', 'DD-MM-YY'), PSR.MON_PTS,
  TO_DATE('18-12-12', 'DD-MM-YY'), PSR.TUE_PTS,
  TO_DATE('19-12-12', 'DD-MM-YY'), PSR.WED_PTS,
  TO_DATE('20-12-12', 'DD-MM-YY'), PSR.THU_PTS,
  TO_DATE('21-12-12', 'DD-MM-YY'), PSR.FRI_PTS,
  TO_DATE('22-12-12', 'DD-MM-YY'), PSR.SAT_PTS,
  TO_DATE('23-12-12', 'DD-MM-YY'), PSR.SUN_PTS) AS POINTS
FROM
  TPOINTSTATEREPORT PSR
JOIN TREGISTEREDDATES RD
ON
  1 = 1
  WHERE RD.REGISTEREDDATE IN (
  TO_DATE('17-12-12', 'DD-MM-YY'),
  TO_DATE('18-12-12', 'DD-MM-YY'),
  TO_DATE('19-12-12', 'DD-MM-YY'),
  TO_DATE('20-12-12', 'DD-MM-YY'),
  TO_DATE('21-12-12', 'DD-MM-YY'),
  TO_DATE('22-12-12', 'DD-MM-YY'),
  TO_DATE('23-12-12', 'DD-MM-YY'));
```

Predefiniowany operator okazał się szybszy w testach wydajnościowych. Uzyskane czasy wykonania znajdują się w tabeli 16.

Zapytanie	**Czas pierwszego wykonania [s]**	**Czas kolejnych wykonań [s]**
Z operatorem *UNPIVOT*	0,005	0,002
Bez operatora *UNPIVOT*	0,014	0,004

Tabela 16. Porównanie wydajności dwóch zapytań, bez i z operatorem *UNPIVOT*.

4.3. Usprawnienia w instrukcjach *DDL*

Z każdą nową wersją, wprowadza się wiele niewielkich usprawnień, które pozwalają jednak na znaczącą poprawę efektywności. Są one szczególnie przydane, gdy należy wykonać zmiany w środowisku produkcyjnym, które musi być dostępne cały czas, bez uszczerbku na wydajności. Z tego też powodu należy o nich wspomnieć.

4.3.1. Ulepszone ograniczenie *NOT NULL*

Znacząco poprawiono wydajność ograniczenia *NOT NULL*, co ma ogromne znaczenie przy wielkich tabelach. We wcześniejszych wersjach *Oracle Database*, dodanie kolumny z ograniczeniem *NOT NULL* i domyślną wartością powodowało masową aktualizację wszystkich istniejących wierszy w tabeli [12]. Każda operacja była traktowana jak zwykła operacja aktualizacji wykonana przez użytkownika, rejestrowana przez dziennik powtórzeń a ostatecznie archiwizowana. Mechanizm ten został

zoptymalizowany. Dodanie ograniczenia *NOT NULL* lub utworzenie nowej kolumny z tymże ograniczeniem powoduje jedynie utworzenie wpisu w metadanych tabeli. SZDB dzięki tej informacji wie, które rekordy mają logicznie ustawioną domyślną wartość kolumny.

4.3.2. Automatyczne oczekiwanie na zdjęcie blokady

Obiekty słownikowe w bazie danych mogą zostać zmodyfikowane tylko wtedy, kiedy żadna sesja nie posiada dla nich wyłącznej blokady do zapisu. W przypadku systemów o wysokiej zajętości wprowadzenie zmian jest prawie niemożliwe. Próba realizacji zmiany kończy się w większości przypadków błędem informującym o nałożonej blokadzie. Należy powtarzać czynność i liczyć na to, że uda trafić się w okno czasowe, w którym obiekt nie jest zajęty. Niedogodność ta została rozwiązana poprzez wprowadzenie dynamicznego parametru instancji bazy danych. Nazywa się on *DDL_LOCK_TIMEOUT* i jest wyrażony w sekundach [13]. Prawidłowe wartości należą do zbioru [0, 1000000]. Najmniejsza, równa 0, oznacza, że sesja nie będzie oczekiwać na zwolnienie blokad z obiektu w celu wykonania instrukcji *DDL*. Największa, równa 1000000, oznacza, że sesja będzie czekać przez nieograniczony czas. Gdy pojawi się możliwość, obiekt zostanie zmodyfikowany.

Aby sprawdzić działanie otworzono dwie sesje. W pierwszej wykonano aktualizację wszystkich wierszy w tabeli *TPOINTSTATE* za pomocą poniższej instrukcji tak, aby założyć na niej blokadę.

```
UPDATE TPOINTSTATE SET POINTS = 1000;
```

Wartość parametru może być modyfikowana w dowolnym momencie w obrębie instancji lub sesji.

```
ALTER SESSION SET DDL_LOCK_TIMEOUT = 10;
```

Sprawdzono zachowanie czterech instrukcji *DDL* modyfikujących strukturę, dla dwóch wartości parametru *DDL_LOCK_TIMEOUT*. Dla wartości 0, oczekiwanym zachowaniem jest natychmiastowe zwrócenie błędu „*ORA-00054: resource busy and acquire with NOWAIT specified*”. Dla wartości 10, oczekiwanym zachowaniem będzie wykonanie instrukcji, jeśli w ciągu 10 sekund nastąpi zwolnienie blokady. W przeciwnym przypadku wystąpi błąd „*ORA-00054: resource busy and acquire with NOWAIT specified*”. Rezultaty zostały umieszczone w tabeli 17. Zaskoczeniem jest działanie instrukcji dodającej nową kolumnę. Bez względu na wartość parametru

DDL_LOCK_TIMEOUT, sesja oczekuje na zwolnienie blokady. Literatura nie podaje informacji świadczącej o tym, że instrukcja służąca do dodania nowych kolumn powinna zostać obsłużona inaczej niż reszta operacji *DDL* dla tabeli.

Instrukcja	**DDL_LOCK_TIMEOUT=0**	**DDL_LOCK_TIMEOUT=10**
ALTER TABLE TPOINTSTATE RENAME TO TEST;	Błąd *ORA-00054*	Wykonanie polecenia, jeśli blokada zwolniła się w ciągu 10 sekund. W przeciwnym przypadku, błąd *ORA-00054*.
DROP TABLE TPOINTSTATE;	Błąd *ORA-00054*	Wykonanie polecenia, jeśli blokada zwolniła się w ciągu 10 sekund. W przeciwnym przypadku, błąd *ORA-00054*.
ALTER TABLE TPOINTSTATE DROP COLUMN POINTS;	Błąd *ORA-00054*	Wykonanie polecenia, jeśli blokada zwolniła się w ciągu 10 sekund. W przeciwnym przypadku, błąd *ORA-00054*.
ALTER TABLE TPOINTSTATE ADD TEST NUMBER;	Oczekiwanie na zwolnienie blokady przez nieograniczony czas. Wykonanie polecenia po zwolnieniu blokady.	Oczekiwanie na zwolnienie blokady przez nieograniczony czas. Wykonanie polecenia po zwolnieniu blokady.
ALTER TABLE TPOINTSTATE MODIFY POINTSTATETYPEID NUMBER(38);	Błąd *ORA-00054*	Wykonanie polecenia, jeśli blokada zwolniła się w ciągu 10 sekund. W przeciwnym przypadku, błąd *ORA-00054*.

Tabela 17. Zachowanie poleceń *DDL* dla różnych wartości *DDL_LOCK_TIMEOUT*.

4.3.3. Tabele tylko do odczytu

Wprowadzono nową klauzulę w instrukcji *ALTER*, która pozwala na zmianę stanu tabeli pomiędzy *READ WRITE* i *READ ONLY* [10]. Umożliwia ona na przejście tabeli w tryb tylko do odczytu lub do odczytu i zapisu. Tryb *READ ONLY* zapobiega wykonaniu instrukcji *DML* na tabeli. Opcja jest przydatna wtedy, gdy obiekt musi być do wyłącznej dyspozycji administratora w celach konserwacyjnych. Aby zmienić tryb tabeli na tylko do odczytu skorzystano z poniższego polecenia.

```
ALTER TABLE  TPOINTSTATE READ ONLY;
```

W widoku słownikowym *[ALL|DBA|USER]_TABLES* znajduje się nowa kolumna, nazwana *READ_ONLY*, która informuje o bieżącym statusie. Jeśli tabela jest w

stanie tylko do odczytu to wartość jest równa *YES*. W przeciwnym przypadku, wartość jest równa „*NO*”. Próba wykonania instrukcji *INSERT*, *UPDATE* lub *DELETE* dla tabeli tylko do odczytu powoduje pojawienie się komunikatu „*ORA-12081: update operation not allowed on table "SCHEMA"."TABLE_NAME".* Opcja nie ma wpływu na wykonywanie zapytań. Aby powrócić do trybu odczytu i zapisu należy wykonać poniższe polecenie.

```
ALTER TABLE  TPOINTSTATE READ WRITE;
```

We wcześniejszych wersjach ten sam efekt był możliwy do osiągniecia poprzez nadanie wszystkim użytkownikom jedynie uprawnienia *SELECT* dla tabeli. Rozwiązanie to nie pozwalało jednak zabronić dostępu do tabeli właścicielowi. Istnieje możliwość odrzucania sesji właściciela obiektu za pomocą wyzwalacza schematu, co spowoduje, że tabela będzie w pełni zabezpieczona przed zmianą. Inną opcją było stworzenie wyzwalacza *DML*, który zgłasza błąd w przypadku wystąpienia operacji *INSERT*, *DELETE* lub *UPDATE*. Oba te rozwiązania mogą jednak powodować zamieszanie ze względu na nieoczekiwany rezultat dla użytkownika. Ponadto zaproponowane obejścia mają negatywny wpływ na wydajność, ponieważ operacje *DML* są blokowane dopiero po przełączeniu w kontekst *PL/SQL* a nie bezpośrednio w *SQL*, jak ma to miejsce po zastosowaniu nowej i wygodnej funkcjonalności.

4.3.4. Niewidzialny indeks

Każdy indeks, po zakończeniu budowy, staje się widoczny dla SZBD i optymalizatora. Może to spowodować gwałtowną zmianę planu wykonania wszystkich zapytań realizowanych w bazie danych. Zmiana może nieść ze sobą pozytywne jak i negatywne skutki. Budowa indeksu, w przypadku dużych tabel, może trwać kilka godzin. Tym bardziej jest to prawdopodobne, im bardziej zróżnicowany rozkład ma kolumna, dla której należy zbudować indeks. Chęć przetestowania wydajności bez wykorzystania istniejącego indeksu wymuszała na administratorze usunięcie indeksu a w razie regresu, czasochłonną odbudowę [12]. Nowa funkcjonalność pozwala na utworzenie indeksów, które staną się widoczne dla optymalizatora dopiero po jawnym przejściu w widoczny stan. Można także przełączyć aktualnie wykorzystywany indeks w stan niewidoczny, co spowoduje unieważnienie wszystkich planów zapytań, które korzystają z niego. Nowe plany zostaną stworzone w oparciu o szacowania optymalizatora, który podczas analizy nie uwzględni niewidzialnego indeksu. Szczególną

uwagę należy zwrócić na wartość parametru systemowego *OPTIMIZER_USE_INVISIBLE_INDEXES*. Musi być ona równa *FALSE*. Jeśli tak nie będzie to optymalizator uwzględni niewidoczne indeksy. Poniższe zapytanie służy do sprawdzenia czy w danej chwili istnieje sojusz o podanej wartości identyfikatora zewnętrznego. Jest ono uruchamiane ponad 20 tysięcy razy podczas ładowania stanów punktowych z danego znacznika czasowego dla danego świata.

```
SELECT
  ACCOUNTCID
FROM
  TACCOUNT
WHERE
  EXTERNALID   = 1000
AND ACCOUNTCID > 0
AND TO_DATE('121218.00', 'YYMMDD.HH24') BETWEEN FROMDATE AND
NVL(TODATE, TO_DATE('121218.00', 'YYMMDD.HH24'));
```

Aktualny plan wykonania jest przedstawiony na rysunku 3. Indeks dla kolumny *EXTERNALID* jest bardzo skuteczny, ponieważ wartości są niemal unikalne.

OPERATION	OBJECT_NAME	OPTIONS	COST
SELECT STATEMENT			1
TABLE ACCESS	TACCOUNT	BY INDEX ROWID	1
Filter Predicates			
INDEX	TACCOUNTEXTERNALID	RANGE SCAN	1

Rysunek 3. Plan wykonania zapytania dla widocznego indeksu.

Aby indeks stał się niewidoczny i pomijany przez optymalizator należy skorzystać z poniższej instrukcji modyfikującej.

```
ALTER INDEX TACCOUNTEXTERNALID INVISIBLE;
```

Plan wykonania zapytania zmienił się na niekorzyść. Jest on przedstawiony na rysunku 4. Zamiast indeksu został wykorzystany pełny odczyt tabeli. Informacja o statusie znajduje się w widoku słownikowych [*ALL|DBA|USER]_INDEXES*, w kolumnie nazwanej *VISIBILITY*.

OPERATION	OBJECT_NAME	OPTIONS	COST
SELECT STATEMENT			9
TABLE ACCESS	TACCOUNT	FULL	9

Rysunek 4. Plan wykonania zapytania dla niewidocznego indeksu.

Aby indeks stał się znowu widoczny należy użyć tej samej instrukcji zastępując jedynie słowo kluczowe *INVISIBLE* słowem *VISIBLE*.

```
ALTER INDEX TACCOUNTEXTERNALID VISIBLE;
```

Najczęściej, cecha przyda się podczas dodawania nowych indeksów. Polecenie tworzące niewidoczny indeks może zostać dodane do harmonogramu i zakończone bez asysty administratora. Indeks nie zostanie użyty do czasu uwidocznienia. Wpływ indeksu na wydajność może zostać zbadany w dogodnej porze.

```
CREATE
  INDEX TACCOUNTEXTERNALUNIVERSEID ON TACCOUNT
  (
    EXTERNALID,
    UNIVERSEID
  ) INVISIBLE;
```

4.4. Zarządzanie statystykami

Nowe statystyki, podobnie jak indeks dla tabeli, mogą wpłynąć dwojako na wydajność instancji bazy danych. Z tego też powodu w ręce projektantów i administratorów oddano narzędzia na zarządzania stanem zgromadzonych statystyk oraz możliwość szybkiego przywracania statystyk do wersji, która jest przechowywana w repozytorium.

4.4.1. Stany statystyk

Zebrane statystyki mogą znajdować się w stanie oczekiwania lub być opublikowane. Domyślnie, każde nowo zebrane statystyki są publikowane automatycznie. Własność tę można sprawdzić poprzez wykonanie funkcji *GET_PREFS* z pakietu *DBMS_STAT* [14]. W tymże pakiecie znajduje się cały mechanizm do zarządzania statystykami wraz z interfejsem. Przyjmuje ona trzy argumenty, z czego dwa ostatnie są domyślnie równe *NULL*. Rezultat *TRUE*, oznacza, że statystyki są wdrażane natychmiast po zebraniu. Wartość parametru dotyczy globalnie całej bazy danych, jako, że pominięte argumenty, oznaczają odpowiednio nazwę schematu i nazwę tabeli.

```
SELECT DBMS_STATS.GET_PREFS('PUBLISH') FROM DUAL;
```

Automatyczna publikacja nie jest najlepszym rozwiązaniem, ponieważ pozbawia administratora wiedzy na temat tego, co mogło spowodować spadek wydajności. Lepszym sposobem jest manualna i świadoma publikacja, która pozwoli administratorowi błyskawicznie zareagować. Aby zmienić zachowanie systemu należy upewnić się, że systemowy parametr *OPTIMIZER_USE_PENDING_STATISTICS* jest ustawiony na wartość *FALSE*. W przeciwnym przypadku nawet nieopublikowane

statystyki będą widoczne dla optymalizatora, zakładając, że są one nowsze niż ostatnie opublikowane statystyki [13]. Automatyczną publikację statystyk można modyfikować w kontekście systemu, schematu lub tabeli. W tym celu należy skorzystać odpowiednio z procedury *SET_GLOBAL_PREFS*, *SET_SCHEMA_PREFS* lub *SET_TABLE_PREFS*. Aby zmienić globalne ustawienia, wymagana jest rola *SYSDBA*.

```
EXECUTE DBMS_STATS.SET_GLOBAL_PREFS ('PUBLISH', 'FALSE');
```

4.4.2. Publikacja statystyk

Informacje o statystkach w stanie oczekiwania znajdują się w widokach słownikowych opisanych w tabeli 18.

Nazwa widoku	**Zawartość**
[ALL|DBA|USER]_TAB_PENDING_STATS	Oczekujące statystyki tabel.
[ALL|DBA|USER]_COL_PENDING_STATS	Oczekujące statystyki kolumn dla tabel.
[ALL|DBA|USER]_IND_PENDING_STATS	Oczekujące statystyki indeksów dla kolumn
[ALL|DBA|USER]_TAB_HISTGRM_PENDING_STATS	Oczekujące statystyki w postaci histogramów dla kolumn w tabelach.

Tabela 18. Widoki słownikowe przechowujące oczekujące statystyki

Najczęściej statystyki gromadzi się dla pojedynczych schematów. W celu wygenerowania statystyk dla schematu aplikacji testowej, uruchomiono procedurę *GATHER_SCHEMA_STATS*.

```
EXECUTE DBMS_STATS.GATHER_SCHEMA_STATS('OIMS');
```

Rezultatem są wpisy, które pojawiły się w widokach systemowych dla wszystkich tabel. Można je uzyskać wykonując zapytania do słownika z narzuconym schematem w warunku.

```
SELECT * FROM DBA_TAB_PENDING_STATS WHERE OWNER = 'OIMS';
SELECT * FROM DBA_COL_PENDING_STATS WHERE OWNER = 'OIMS';
SELECT * FROM DBA_IND_PENDING_STATS WHERE OWNER = 'OIMS';
SELECT * FROM DBA_TAB_HISTGRM_PENDING_STATS WHERE OWNER = 'OIMS';
```

Statystyki w stanie oczekiwania mogą zostać usunięte za pomocą procedury *DELETE_PENDING_STATS*. Przyjmuje ona dwa parametry, które domyślnie są wartościami *NULL*. Pierwszy pozwala na zdefiniowanie schematu a drugi obiektu, którego statystyki oczekujące mają zostać skasowane. W przypadku braku argumentów, skasowane zostaną wpisy na poziomie całej bazy danych.

```
EXECUTE DBMS_STATS.DELETE_PENDING_STATS('OIMS');
```

Najczęściej, administrator chce opublikować i przetestować statystyki w stanie oczekiwania. Do tego celu służy procedura *PUBLISH_PENDING_STATS*. Posiada dokładnie te same argumenty, co procedura *DELETE_PENDING_STATS*. Różnica polega na tym, że przy publikacji należy jawnie przekazać dwa parametry, ponieważ w prototypie nie są one domyślnie ustawiane, jako *NULL*. W przeciwnym przypadku zostanie zgłoszony błąd o zbyt małej liczbie przekazanych argumentów. Po wykonaniu operacji, przetworzone wpisy znikają z widoków statystyk w stanie oczekiwania.

```
EXECUTE DBMS_STATS.PUBLISH_PENDING_STATS('OIMS', NULL);
```

4.4.3. Repozytorium starych statystyk

Mechanizm oczekiwania i publikacji statystyk byłby bezużyteczny gdyby w ręce administratora nie zostało przekazane narzędzie pozwalające na łatwe i szybkie przechowywanie oraz przywracanie starych statystyk. Taka konieczność może wystąpić, jeśli nowe statystyki spowodowały spadek wydajności. We wcześniejszych wersjach oprogramowania, należało ręcznie zrobić kopię starych statystyk, przed uruchomieniem zbierania nowych. Jeśli kopia nie została utworzona, starych statystyk nie dało się odtworzyć. Domyślnie repozytorium przechowuje statystyki z ostatnich 31 dni. Można to zweryfikować poprzez uruchomienie funkcji *GET_STATS_HISTORY_RETENTION*. Jeśli nie wprowadzano wcześniej zmian w konfiguracji to rezultatem będzie liczba 31. Starsze wpisy będą automatycznie usuwane.

```
SELECT DBMS_STATS.GET_STATS_HISTORY_RETENTION() FROM DUAL;
```

Ponadto istnieje możliwość sprawdzenia, na jaki znacznik czasowy wskazuje najstarszy wpis.

```
SELECT DBMS_STATS.GET_STATS_HISTORY_AVAILABILITY() FROM DUAL;
```

Długość historii można zmienić za pomocą procedury *ALTER_STATS_HISTORY_RETENTION*. Przyjmuje ona jeden parametr całkowitoliczbowy, który wyraża w dniach jak długa historia statystyk jest przechowywana w repozytorium. Dozwolone wartości należą do przedziału [1,365000] [14]. Ponadto, szczególne wartości mają specjalne znaczenie:

- Wartość 0 oznacza, że historia statystyk nie jest przechowywana.

- Wartość 1 oznacza, że historia nie będzie automatycznie obcinana. Skutkiem tego jest przechowywanie wpisów historycznych przez nieograniczony czas o ile nie zostaną one ręcznie usunięte lub nie zabraknie zasobów do przechowywania historii.
- Wartość *NULL*, informuje procedurę, że należy przywrócić wartość domyślną parametru tj. 31.

Zawartość repozytorium można przyciąć ręcznie za pomocą procedury *PURGE_STATS*, która przyjmuje jako parametr znacznik czasowy. Wszystkie wpisy historyczne, które pochodzą sprzed podanego czasu zostaną usunięte. W tym przypadku zostaną usunięte wszystkie statystyki starsze niż 10 dni.

```
EXECUTE DBMS_STATS.PURGE_STATS(SYSTIMESTAMP - 10);
```

4.4.4. Przywracanie statystyk

Aby przywrócić stare statystyki, w przypadku negatywnych zmian, należy odnaleźć poprawnie działający wpis w repozytorium. Można tego dokonać poprzez wykonanie zapytania *DBA_OPTSTAT_OPERATIONS*. Przechowywane są w nim zdarzenia związane ze zbieraniem statystyk. Nowe wpisy nie powstają, jeśli repozytorium historii jest całkowicie wyłączone.

```
SELECT * FROM DBA_OPTSTAT_OPERATIONS WHERE TARGET = 'OIMS';
```

Jeśli mają to być ostatnio działające statystyki to wystarczy bez dokładnego sprawdzania przyjąć, że należy cofnąć stan statystyk do czasu sprzed aktualizacji. Może to być np. stan sprzed 24 godzin. Do przywracania służą procedury wymienione w tabeli 19.

Procedura	**Opis**
RESTORE_SCHEMA_STATS	Przywraca statystyki wszystkich tabel z danego schematu.
RESTORE_SYSTEM_STATS	Przywraca statystyki systemowe.
RESTORE_TABLE_STATS	Przywraca statystyki dla danej tabeli.
RESTORE_DICTIONARY_STATS	Przywraca statystyki dla tabel słownikowych tzn. takich, które należą m.in. do schematów *SYS* oraz *SYSTEM*.
RESTORE_FIXED_OBJECTS_STATS	Przywraca statystyki tabel stałych. Są to obiekty, których nazwy zaczynają się od ciągu znaków „X$”.
RESTORE_DATABASE_STATS	Przywraca statystyki dla całej bazy danych.

Tabela 19. Procedury służące do przywracania statystyk.

Aby przywrócić statystyki schematu do stanu sprzed 24 godzin należy skorzystać z procedury *RESTORE_SCHEMA_STATS*. Posiada ona cztery parametry:

- Nazwa schematu, którego statystyki należy przywrócić.
- Znacznik czasu, do którego należy przywrócić statystyki.
- Flaga wymuszająca przywrócenie nawet, jeśli statystyki są zablokowane. Domyślnie *FALSE*.
- Flaga pomijająca unieważnienie kursorów zależnych od planów wykonania, które zostały wyznaczone w oparciu o poprzednie statystyki. Domyślnie zachowanie zależy od wartości parametru *NO_INVALIDATE*.

```
EXECUTE DBMS_STATS.RESTORE_SCHEMA_STATS('OIMS', SYSTIMESTAMP - 1);
```

Każde opublikowanie i przywrócenie statystyk powoduje pojawienie się wpisu w widoku *DBA_TAB_STATS_HISTORY*, dla każdej tabeli.

```
SELECT * FROM DBA_TAB_STATS_HISTORY WHERE OWNER = 'OIMS';
```

4.5. Partycjonowanie

Partycjonowanie jest jedną z najważniejszych cech nowoczesnych baz danych. Okazało się przełomem, gdy zostało wprowadzone wraz z *Oracle Database 8* w 1997 roku. Polega ono na podziale dużych obiektów na mniejsze części. Tabela lub indeks, są widoczne przez użytkownika, jako logiczna całość, podczas gdy fizycznie mogą być one podzielone na wiele fragmentów, którymi zarządza SZBD. Nierzadko podział fizyczny bazy danych odpowiada podziałowi logicznemu organizacji, która korzysta z niej. Mechanizm pozwala w znaczący sposób zwiększyć ogólną wydajność systemu, co jest spowodowane krótszym czasem dostępu do danych. Blokady wyłączne są zakładane jedynie na fragmenty obiektów. Powoduje to mniejszą rywalizację o zasoby. Poszczególne jednostki organizacji mogą potrzebować zupełnie różnych fragmentów tabeli, co pozwala na całkowite wyeliminowanie kolizji.

Istnieje wiele rodzajów partycjonowania obiektów. Każdy z nich ma swoje specyficzne zastosowanie. Opisy oraz wskazówki, jakimi należy się kierować podczas wyboru rodzaju partycjonowania, znajdują się w literaturze [15]. Mechanizm jest ciągle rozwijany, o czym można się przekonać w tymże rozdziale.

4.5.1. Partycjonowanie przedziałowe

Dotychczas, w partycjonowaniu zakresowym, tworzenie partycji należało do zadań administratora. Jest to uciążliwe, jeśli tabele partycjonuje się według kolumny zawierającej datę a poszczególne partycje są tworzone co miesiąc. Znane są też przypadki partycjonowania dziennego. Jedyną możliwością utrzymania tabeli jest stworzenie zadania w harmonogramie, które codziennie wyręcza administratora z tego żmudnego zadania. W przypadku braku partycji odpowiedniej dla rekordu, pojawiał się błąd podczas operacji *DML*.

Rozwiązaniem problemu jest partycjonowanie przedziałowe. Wymagane partycje zostaną utworzone automatycznie przez SZDB w momencie, gdy stanie się to konieczne. Aby skorzystać z tej funkcjonalności należy podczas tworzenia tabeli zdefiniować, co najmniej jedną partycję, która zostanie punktem przejścia. Na podstawie punktu przejścia oraz interwału, SZDB będzie wstanie utworzyć kolejną wymaganą partycję. Mechanizm posiada pewne ograniczenia [16]:

- Tabela może być partycjonowana tylko względem jednej kolumny.
- Klucz partycjonowania musi być typu *DATE* lub *NUMBER*.
- Nie można partycjonować przedziałowo tabeli indeksowej.
- Dla tabeli partycjonowanej przedziałowo nie można utworzyć indeksu domenowego.
- Nie można tworzyć podpartycji przedziałowych dla partycji przedziałowej. Oznacza to, że nie istnieje możliwość skorzystania ze scenariusza „przedziałowo-przedziałowe” w przypadku partycjonowania złożonego.

Tabela *TPOINSTATE*, przechowująca stany punktowe, jest bardzo trudna w utrzymaniu ze względu na planowaną dużą liczbę wierszy. Aby zachować wydajność na względnie stałym poziomie należy wykorzystać partycjonowanie. Indeksy mogą być mniej skuteczne niż zwykle lub zupełnie pomijane ze względu na niewielką unikalność wartości (*RANGE SCAN*). Poniżej znajduje się jedno z najczęściej wykonywanych zapytań odpowiadające za zwrócenie stanów punktowych z danego dnia dla wszystkich członków jednego sojuszu.

```
SELECT * FROM TPOINTSTATE WHERE CHECKPOINTDATE = TO_DATE('17-12-12
00:00:00') AND POINTSTATETYPEID = 1 AND ALLIANCECID = 2000;
```

Aktualny plan wykonania znajduje się na rysunku 5. Po podziale, pełne odczyty będą dotyczyć tylko pojedynczych partycji a nie całej tabeli.

Rysunek 5. Plan wykonania zapytania przed partycjonowaniem.

Można dokonać partycjonowania istniejącej tabeli na kilka sposobów. Najbardziej oczywistym rozwiązaniem, wydaje się poniższa sekwencja:

1. Utworzenie nowej tabeli z taką samą definicją jak stara tabela, ale ze zdefiniowanymi partycjami.
2. Wstawienie wierszy ze starej tabeli do nowej tabeli za pomocą instrukcji *INSERT* lub narzędzia do eksportu i importu danych.
3. Usunięcie starej tabeli.
4. Zmiana nazwy nowej tabeli na nazwę usuniętej tabeli.

Opisane rozwiązanie jest ciężkie do wykonania w systemie, który jest cały czas w użyciu. Usunięcie obiektu powoduje dekompilację obiektów zależnych oraz czasowy brak tabeli o wymaganej nazwie. Ponadto operacja wstawienia wszystkich danych ze starej tabeli do nowego obiektu generuje niekontrolowane obciążenie, ponieważ miejsce na nowe rekordy musi zostać zaalokowane. Lepszym rozwiązaniem jest wykorzystanie pakietu *DBMS_REDEFINITION*, który służy do zmiany obiektów online, bez przerwy w działaniu całego systemu oraz z mniejszym uszczerbkiem na wydajności. Wiersze z tabeli źródłowej są przenoszone w bardziej inteligentny sposób a nie poprzez naiwne kopiowanie. Operacja musi zostać wykonana z poziomu użytkownika o uprawnieniach *SYSDBA*.

Pierwszą czynnością, jaką należy wykonać jest sprawdzenie czy istnieje możliwość redefiniowania obiektu online. Jeśli procedura *CAN_REDEF_TABLE* wykona się bez błędów to znaczy, że można redefiniować tabelę.

```
EXECUTE DBMS_REDEFINITION.CAN_REDEF_TABLE('OIMS', 'TPOINTSTATE');
```

Następnie należy stworzyć tabelę o identycznej strukturze jak tabela źródłowa, z pominięciem ograniczeń i indeksów. W definicji muszą zostać zawarte informacje o partycjonowaniu. W tym przypadku zostało użyte partycjonowanie zakresowe.

```
CREATE TABLE OIMS.TPOINTSTATER
     (

        POINTSTATEID  NUMBER(38, 0),
        POINTSTATELEVELID  NUMBER(1, 0),
        POINTSTATETYPEID  NUMBER(2, 0),
        ACCOUNTCID NUMBER(15,0),
        ALLIANCECID NUMBER(5,0),
        CHECKPOINTDATE DATE,
        POINTS NUMBER(10, 0)
      )
PARTITION BY RANGE(CHECKPOINTDATE)
(
PARTITION PPOINTSTATE201212
VALUES LESS THAN (TO_DATE('01-01-2013','DD-MM-YYYY'))
);
```

Po utworzeniu tabeli z nową definicją, konieczne jest zainicjalizowanie procesu redefinicji za pomocą procedury *START_REDEF_TABLE*. Przyjmuje ona trzy parametry:

- Nazwę schematu,
- Nazwę tabeli redefiniowanej,
- Nazwę tabeli z docelową strukturą;

```
EXECUTE DBMS_REDEFINITION.START_REDEF_TABLE('OIMS', 'TPOINTSTATE',
'TPOINTSTATER');
```

Kolejną czynnością jest skopiowanie obiektów zależnych za pomocą procedury *COPY_TABLE_DEPENDENTS*.

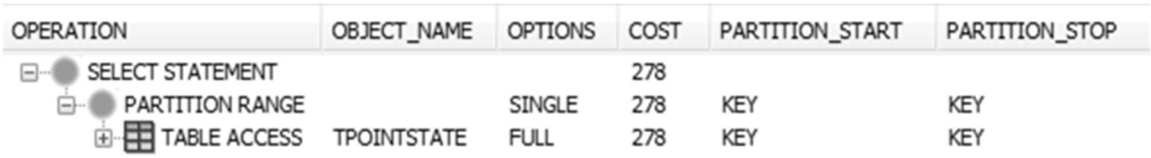
```
DECLARE
  ERROR_COUNT PLS_INTEGER := 0;
BEGIN
  DBMS_REDEFINITION.COPY_TABLE_DEPENDENTS('OIMS', 'TPOINTSTATE',
'TPOINTSTATER'
  , 1, TRUE, TRUE, TRUE, FALSE, ERROR_COUNT);
  DBMS_OUTPUT.PUT_LINE('Errors: ' || TO_CHAR(ERROR_COUNT));
END;
```

Po zakończeniu kopiowania obiektów zależnych, zakończono operację redefiniowania tabeli poprzez wywołanie procedury *FINISH_REDEF_TABLE*. Po jej prawidłowym wykonaniu, oryginalna tabela posiada nową strukturę. Opcjonalnym zadaniem jest zmiana nazw skopiowanych indeksów oraz ograniczeń, które posiadają nazwy wygenerowane przez SZBD.

```
EXECUTE DBMS_REDEFINITION.FINISH_REDEF_TABLE('OIMS', 'TPOINTSTATE',
'TPOINTSTATER');
```

Aby sprawdzić czy istnieją partycje, wystarczy wykonać instrukcję *SELECT*. W planie wykonania zapytania pojawiła się operacja *PARTITION RANGE*, widoczna na

rysunku 6. Nie spowodowało to jednak poprawy wydajności, ponieważ tabela źródłowa znajduje się w początkowym cyklu życia i zawiera dane z krótkiego okresu.



OPERATION	OBJECT_NAME	OPTIONS	COST	PARTITION_START	PARTITION_STOP
SELECT STATEMENT			278		
PARTITION RANGE		SINGLE	278	KEY	KEY
TABLE ACCESS	TPOINTSTATE	FULL	278	KEY	KEY

Rysunek 6. Plan wykonania zapytania po partycjonowaniu zakresowym.

Rezultatem jest tabela partycjonowana posiadająca jedną partycję dla rekordów z wartością klucza partycjonowania wcześniejszą niż 01-01-13 00:00:00. Użyto partycjonowania zakresowego więc wstawienie rekordu z datą późniejszą od 31-12-12 23:59:59, powoduje błąd „*ORA-14400: inserted partition key does not map to any partition*”.

```
INSERT
INTO
  TPOINTSTATE VALUES
  (
    10000000,
    1,
    1,
    5623,
    1049,
    TO_DATE('01-01-13 00:00:00'),
    999
  );
```

Tabela partycjonowana zakresowo może być w łatwy sposób zmieniona na tabelę partycjonowaną przedziałowo za pomocą instrukcji *ALTER*. Zmiana nie ingeruje w strukturę tabeli a jedynie informuje SZBD o tym, że należy tworzyć partycję zgodnie z narzuconym interwałem.

```
ALTER TABLE TPOINTSTATE SET INTERVAL(NUMTOYMINTERVAL(1, 'MONTH'));
```

Ponowne wykonanie instrukcji *INSERT* powiedzie się. W rezultacie zostanie utworzona automatycznie nowa partycja dla rekordów późniejszych niż 31-12-12 23: 59:59 ale wcześniejszych od 01-02-13 00:00:00. Partycje utworzone przez transakcję, która została wycofana, pozostaną puste. Instrukcje *DDL* nie są rejestrowane przez przestrzeń wycofania. Jedyną wadą partycjonowania przedziałowego jest brak możliwości zdefiniowania szablonu nazewnictwa dla nowotworzonych partycji. Tę niedogodność można zaobserwować w tabeli 20. Nazwy są nadawane przez SZBD.

```
SELECT
  PARTITION_NAME,
  HIGH_VALUE
FROM
  USER_TAB_PARTITIONS
WHERE
  TABLE_NAME = 'TPOINTSTATE';
```

PARTITION_NAME	HIGH_VALUE
PPOINTSTATE201212	TO_DATE(' 2013-01-01 00:00:00', 'SYYYY-MM-DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIAN')
SYS_P21	TO_DATE(' 2013-02-01 00:00:00', 'SYYYY-MM-DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIAN')

Tabela 20. Widok partycji zakresowych dla tabeli *TPOINTSTATE*.

Nie jest to uciążliwe, ponieważ nie trzeba operować nazwami partycji w instrukcji *SELECT*, wskazując jawnie skąd rekordy mają zostać pobrane. Klauzula *PARTITION FOR* znajdzie odpowiednią partycję na podstawie przekazanej wartości klucza partycjonowania. Ważne jest, aby przy operowaniu partycjami korzystać z formatowania daty, które wskazuje tysiąclecie. W innym przypadku, tworzenie partycji nie powiedzie się bądź też partycja dla podanej daty nie zostanie znaleziona.

```
SELECT
  *
FROM
  TPOINTSTATE PARTITION FOR (TO_DATE('01-01-2013', 'DD-MM-YYYY'));
```

4.5.2. Partycjonowanie złożone

Partycjonowanie złożone polega na zwiększeniu poziomu ziarnistości obiektu poprzez podział partycji na jeszcze mniejsze podpartycje. Takowa funkcjonalność jest już dostępna od wcześniejszych wersji, ale możliwe było stworzenie tylko dwóch scenariuszy partycjonowania złożonego [15]:

- **Zakresowo-haszowe**, wprowadzone w wersji 8i [2]. Pozwala na partycjonowanie zakresowe a następnie dla każdej partycji tworzone są podpartycje haszowe.
- **Zakresowo-listowe**, wprowadzone w wersji 9i [17]. Pozwala na partycjonowanie zakresowe a następnie dla każdej partycji tworzone są podpartycje listowe.

Wraz z nowszymi wersjami dodano kolejne możliwe konfiguracje partycjonowania złożonego [5]:

- **Zakresowo-zakresowe**, w którym tabela jest partycjonowana zakresowo a następnie dla każdej partycji tworzone są podpartycje zakresowe.
- **Listowo-zakresowe**, w którym tabela jest partycjonowana listowo a następnie dla każdej partycji tworzone są podpartycje zakresowe.
- **Listowo-haszowe**, w którym tabela jest partycjonowana listowo a następnie dla każdej partycji tworzone są podpartycje haszowe.
- **Listowo-listowe**, w którym tabela jest partycjonowana listowo a następnie dla każdej partycji tworzone są podpartycje listowe.
- **Przedziałowo-zakresowe**, w którym tabela jest partycjonowana przedziałowo a następnie dla każdej partycji tworzone są podpartycje zakresowe.
- **Przedziałowo-haszowe**, w którym tabela jest partycjonowana przedziałowo a następnie dla każdej partycji tworzone są podpartycje haszowe.
- **Przedziałowo-listowe**, w którym tabela jest partycjonowana przedziałowo a następnie dla każdej partycji tworzone są podpartycje listowe.

Nowe schematy pozwalają na tworzenie partycji i podpartycji według dwóch różnych kluczy przechowujących datę. W takim przypadku dobrze sprawdzi się partycjonowanie przedziałowo-zakresowe lub zakresowo-zakresowe. Pozostałe konfiguracje, choć na pewno są rzadziej wykorzystywane, dopełniają funkcjonalność dając ogromne możliwości projektantom struktury bazy danych.

Zakłada się, że partycjonowanie złożone przedziałowo-listowe wpłynie pozytywnie na wydajność zapytań odwołujących się do tabeli *TPOINSTATE*. Będzie ona najpierw poddana partycjonowaniu względem daty stanu punktowego a następnie dla każdej partycji zostaną utworzone podpartycje przechowujące stany punktowe poszczególnych typów. Wybór kolumny przechowującej typ stanu punktowego, jako klucz partycjonowania jest lepszym rozwiązaniem niż wybór kolumny przechowującej poziom stanu punktowego. Pomiędzy ilością kont a sojuszy występuje duża dysproporcja. Podpartycja przechowująca stany punktowe dla sojuszy byłaby znacznie mniejsza od podpartycji przechowującej stany punktowe dla graczy. Liczbę wierszy dla obu poziomów stanu punktowego są zawarte w tabeli 21.

```
SELECT
  PSL.NAME,
  COUNT(*) AS QUANTITY
FROM
  TPOINTSTATE PS
JOIN TPOINTSTATELEVEL PSL
ON
  PSL.POINTSTATELEVELID = PS.POINTSTATELEVELID
GROUP BY
  PSL.NAME;
```

NAME	QUANTITY
ACCOUNT	175440
ALLIANCE	29472

Tabela 21. Rozkład stanów punktowych ze względu na poziom.

Rozkład stanów punktowych różnego typu jest zawsze równomierny. W innym przypadku, baza nie będzie w stanie integralności. Załadowanie danych do bazy, po którym ilości stanów punktowych są różne dla poszczególnych typów stanów punktowych, należy uznać za błędne. Oprogramowanie, przed zatwierdzeniem transakcji sprawdza czy integralność jest zachowana. Jeśli nie, transakcja zostaje wycofana. Bieżący rozkład stanów punktowych ze względu na typ jest umieszczony w tabeli 22.

```
SELECT
PST.NAME,
COUNT(*) AS QUANTITY

FROM
TPOINTSTATE PS
JOIN TPOINTSTATETYPE PST
ON
PST.POINTSTATETYPEID = PS.POINTSTATETYPEID
GROUP BY
PST.NAME;
```

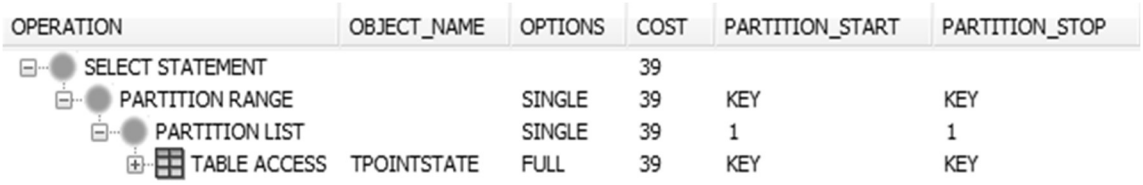
NAME	QUANTITY
DESTROYED_MILITARY	25614
LOST_MILITARY	25614
BUILDED_MILITARY	25614
MILITARY	25614
GENERAL	25614
RESEARCH	25614
ECONOMY	25614
HONOR	25614

Tabela 22. Rozkład stanów punktowych ze względu na typ.

Aby zmodyfikować tabelę *TPOINSTATE*, tak, aby korzystała z partycjonowania złożonego przedziałowo-listowego, należy ją redefiniować z użyciem poniższej struktury.

```
CREATE
  TABLE OIMS.TPOINTSTATER
  (
    POINTSTATEID      NUMBER(38, 0),
    POINTSTATELEVELID NUMBER(1, 0),
    POINTSTATETYPEID  NUMBER(2, 0),
    ACCOUNTCID        NUMBER(15,0),
    ALLIANCECID       NUMBER(5,0),
    CHECKPOINTDATE    DATE,
    POINTS            NUMBER(10, 0)
  )
  PARTITION BY RANGE(CHECKPOINTDATE)
  INTERVAL(NUMTOYMINTERVAL(1, 'MONTH'))
  SUBPARTITION BY LIST (POINTSTATETYPEID)
  SUBPARTITION TEMPLATE
  (
    SUBPARTITION PGENERAL VALUES (1),
    SUBPARTITION PECONOMY VALUES (2),
    SUBPARTITION PRESEARCH VALUES (3),
    SUBPARTITION PMILITARY VALUES (4),
    SUBPARTITION PBMILITARY VALUES (5),
    SUBPARTITION PDMILITARY VALUES (6),
    SUBPARTITION PLMILITARY VALUES(7),
    SUBPARTITION PHONOR VALUES (8)
  )
  (
    PARTITION PPOINTSTATE201212 VALUES LESS THAN (
    TO_DATE('01-01-2013', 'DD-MM-YYYY'))
  );
```

Tak jak zakładano, wprowadzenie partycjonowania złożonego wpłynęło pozytywnie na koszt dostępu do danych. Umożliwia ono zachowanie stałej wydajności przez cały czas eksploatacji tabeli. Koszt będzie wzrastał tylko do pewnej wartości progowej, która jest nieprzekraczalna. Granica ta zostanie osiągnięta po pełnym miesiącu użytkowania. W planie wykonania zapytania, który można zaobserwować na rysunku 7, z odpowiednio wybranej partycji przedziałowej zostaje wybrana podpartycja listowa przechowująca stany punktowe żądanego typu. Po dokonaniu partycjonowania tabeli, która z założenia będzie przechowywać wiele wierszy, można przystąpić do testowania indeksów. Mogą to być indeksy globalne - dla całej tabeli lub lokalne – dla poszczególnych partycji.



Rysunek 7. Plan wykonania zapytania po partycjonowaniu przedziałowo-listowym.

Nowe partycje przedziałowe, wraz z podpartycjami, będą tworzone automatycznie według zdefiniowanego schematu. W tabeli 23 należy zwrócić szczególną uwagę na pole informujące o tym, że partycje są złożone z 8 podpartycji.

```
SELECT
  PARTITION_NAME,
  COMPOSITE,
  SUBPARTITION_COUNT,
  HIGH_VALUE
FROM
  USER_TAB_PARTITIONS
WHERE
  TABLE_NAME = 'TPOINTSTATE';
```

PARTITION_NAME	COMPOSITE	SUBPARTITION_COUNT	HIGH_VALUE
PPOINTSTATE201212	YES	8	TO_DATE(' 2013-01-01 00:00:00', 'SYYYY-MM-DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIAN')
SYS_P69	YES	8	TO_DATE(' 2013-02-01 00:00:00', 'SYYYY-MM-DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIAN')

Tabela 23. Widok partycji złożonych dla tabeli *TPOINTSTATE*.

Informacje o podpartycjach można uzyskać poprzez wykonanie zapytania na widoku [*DBA|USER|ALL*]*_TAB_SUBPARTITIONS*. Tabela 24 pokazuje wszystkie 16 podpartycji.

```
SELECT
  PARTITION_NAME,
  SUBPARTITION_NAME,
  HIGH_VALUE
FROM
  USER_TAB_SUBPARTITIONS
WHERE
  TABLE_NAME = 'TPOINTSTATE';
```

PARTITION_NAME	SUBPARTITION_NAME	HIGH_VALUE
PPOINTSTATE201212	PPOINTSTATE201212_PGENERAL	1
PPOINTSTATE201212	PPOINTSTATE201212_PECONOMY	2
PPOINTSTATE201212	PPOINTSTATE201212_PRESEARCH	3
PPOINTSTATE201212	PPOINTSTATE201212_PMILITARY	4
PPOINTSTATE201212	PPOINTSTATE201212_PBMILITARY	5
PPOINTSTATE201212	PPOINTSTATE201212_PDMILITARY	6
PPOINTSTATE201212	PPOINTSTATE201212_PLMILITARY	7
PPOINTSTATE201212	PPOINTSTATE201212_PHONOR	8
SYS_P69	SYS_SUBP61	1
SYS_P69	SYS_SUBP62	2
SYS_P69	SYS_SUBP63	3
SYS_P69	SYS_SUBP64	4
SYS_P69	SYS_SUBP65	5
SYS_P69	SYS_SUBP66	6
SYS_P69	SYS_SUBP67	7
SYS_P69	SYS_SUBP68	8

Tabela 24. Widok podpartycji dla tabeli *TPOINTSTATE*.

4.5.3. Partycjonowanie referencyjne

Partycjonowanie referencyjne pozwala na równoległe partycjonowanie tabeli podrzędnej zgodnie z zasadami zdefiniowanymi w tabeli nadrzędnej. Funkcjonalność może być wykorzystana tylko wtedy, gdy istnieją dwie tabele będące ze sobą w relacji zdefiniowanej poprzez ograniczenia *FOREIGN KEY* i *PRIMARY* KEY. Kluczem partycjonowania dla tabeli podrzędnej jest kolumna, która należy tylko i wyłącznie do tabeli nadrzędnej. Zapobiega to duplikacji klucza partycjonowania w tabeli podrzędnej [16]. Mechanizm w znaczący sposób ułatwia utrzymanie bazy danych. Tabela podrzędna musi deklarować chęć skorzystania z partycjonowania referencyjnego wskazując nazwę ograniczenia *FOREIGN KEY*. Takowe ograniczenie wskazuje natomiast tabelę nadrzędną

i kolumnę klucza głównego tejże tabeli. Struktura partycji z tabeli nadrzędnej będzie na bieżąco odwzorowywana w tabeli podrzędnej.

Niestety, partycjonowanie referencyjne nie jest obsługiwane, jeśli tabela nadrzędna korzysta z partycjonowania przedziałowego [16]. Przy próbie realizacji takiego scenariusza pojawia się błąd „*ORA-14659: Partitioning method of the parent table is not supported*". Próba partycjonowania w momencie, gdy ograniczenia nie są poprawne powoduje błąd „*ORA-14651: reference partitioning constraint is not supported*". Ograniczenie jest prawidłowe, gdy jest włączone, zweryfikowane i nieodraczalne. Ponadto, wartość klucza obcego nie może być automatycznie zastępowana wartością *NULL* w momencie, gdy wpis z tabeli nadrzędnej zostanie usunięty.

Najważniejszą informacją, która jest generowana przez testową aplikację, są wartości tygodniowych, miesięcznych i rocznych różnic punktowych. Dane te powinny być przechowywane w obliczonej formie, w osobnej tabeli w celu zmniejszenia obciążenia serwera. Służy do tego tabela *TDIFFERENCE*. Przechowuje ona obliczone różnice pomiędzy wszystkimi stanami punktowymi z początku i końca danego okresu dla sojuszy i kont graczy. Okres jest definiowany przez tabelę *TPERIODS*. Jeśli jakiś stan punktowy nie ma swojego odpowiednika to różnica punktowa nie może zostać wyznaczona i nie pojawi się wpis w tabeli. Tabela może być partycjonowana na podstawie obiektu *TPOINSTATE* z zastrzeżeniem, że to stan punktowy z końca okresu będzie determinował przynależność rekordów do partycji w tabeli podrzędnej. Oznacza to, że wpis, który przechowuje różnicę punktową pomiędzy dniami 1.01.2012 i 31.12.2012, znajdzie się w partycji, która odpowiada grudniowym stanom punktowym.

Redefinicja obiektu, tak, aby korzystał z partycjonowania referencyjnego wymaga nadzwyczajnej procedury, która wymaga większego nakładu pracy. Z tego też względu, skasowano ją, utworzono na nowo zgodnie z poniższą strukturą oraz ponownie wypełniono danymi. Pominięto definicję ograniczeń, aby zachować czytelność składni.

```
CREATE
  TABLE TDIFFERENCE
  (
    DIFFERENCEID           NUMBER(38, 0),
    POINTSTATELEVELID      NUMBER(1, 0),
    POINTSTATETYPEID       NUMBER(2, 0),
    ALLIANCECID            NUMBER(5,0),
    ACCOUNTCID             NUMBER(15,0),
    FIRSTPOINTSTATEID      NUMBER(38, 0),
    SECONDPOINTSTATEID     NUMBER(38, 0),
    FIRSTPOINTSTATEVALUE   NUMBER(10, 0),
    SECONDPOINTSTATEVALUE  NUMBER(10, 0),
    PERIODID               NUMBER(5),
    VALUEINUNIT            NUMBER(10, 0),
    VALUEINPERCENT         NUMBER(5, 2)
  )
  PARTITION BY REFERENCE
  (
    CFKDIFFERENCESECONDSTATE
  );
```

Po utworzeniu, tabela *TDIFFERENCE* odziedziczyła automatycznie początkowy schemat partycjonowania tabeli nadrzędnej. W tabeli 25 można zaobserwować partycje istniejące dla obu tabel. Największa dozwolona wartość dla partycji w tabeli podrzędnej jest równa *NULL*, ponieważ to wartość klucza obcego decyduje o tym, do jakiej partycji trafi rekord.

```
SELECT
  TABLE_NAME,
  PARTITION_NAME,
  HIGH_VALUE
FROM
  USER_TAB_PARTITIONS
WHERE
  TABLE_NAME IN ('TPOINTSTATE', 'TDIFFERENCE');
```

TABLE_NAME	PARTITION_NAME	HIGH_VALUE
TPOINTSTATE	PPOINTSTATE201212	TO_DATE(' 2013-01-01 00:00:00', 'SYYYY-MM-DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIAN')
TDIFFERENCE	PPOINTSTATE201212	

Tabela 25. Widok partycji dla tabel powiązanych ograniczeniami.

W widoku [*ALL*|*DBA*|*USER*]_*PART_TABLES* znajdują się informacje o typie partycjonowania oraz nazwie ograniczenia, które jest w tym celu wykorzystane. Wykonując odpowiednie złączenie z widokiem [*ALL*|*DBA*|*USER*]_*CONSTRAINTS* można uzyskać nazwę tabeli nadrzędnej. Rezultat został przedstawiony w tabeli 26.

```
SELECT
  UC2.TABLE_NAME PARENT,
  UPN.PARTITIONING_TYPE,
  UPN.REF_PTN_CONSTRAINT_NAME CONSTRAINT_NAME,
  UPN.TABLE_NAME CHILDREN
FROM
  USER_PART_TABLES UPN
JOIN USER_CONSTRAINTS UC1
ON
  UC1.CONSTRAINT_NAME = UPN.REF_PTN_CONSTRAINT_NAME
JOIN USER_CONSTRAINTS UC2
ON
  UC1.R_CONSTRAINT_NAME = UC2.CONSTRAINT_NAME
WHERE
  UPN.TABLE_NAME IN ('TPOINTSTATE', 'TDIFFERENCE');
```

PARENT	TYPE	CONSTRAINT_NAME	CHILDREN
TPOINTSTATE	REFERENCE	CFKTDIFFERENCESECONDSTATE	TDIFFERENCE

Tabela 26. Informacje o partycjonowaniu, uzyskane ze słownika systemowego.

Utworzono kolejną partycję zakresową dla tabeli nadrzędnej za pomocą poniższego polecenia.

```
ALTER TABLE TPOINTSTATE ADD PARTITION  VALUES LESS THAN
(TO_DATE('2013-02-01', 'YYYY-MM-DD'));
```

Sprawdzono jeszcze raz partycje dla obu skorelowanych tabel. Nowa partycja dla tabeli podrzędnej została automatycznie utworzona. Aktualnie istniejące partycje są wymienione w tabeli 27.

```
SELECT
  TABLE_NAME,
  PARTITION_NAME,
  HIGH_VALUE
FROM
  USER_TAB_PARTITIONS
WHERE
  TABLE_NAME IN ('TPOINTSTATE', 'TDIFFERENCE');
```

TABLE_NAME	PARTITION_NAME	HIGH_VALUE
TPOINTSTATE	PPOINTSTATE201212	TO_DATE(' 2013-01-01 00:00:00', 'SYYYY-MM-DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIAN')
TPOINTSTATE	SYS_P90	TO_DATE(' 2013-02-01 00:00:00', 'SYYYY-MM-DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIAN')
TDIFFERENCE	PPOINTSTATE201212	
TDIFFERENCE	SYS_P91	

Tabela 27. Widok partycji dla tabel powiązanych ograniczeniami po modyfikacji.

4.5.4. Partycjonowanie systemowe

W *Oracle Database* do wersji 11g, wymagane było istnienie klucza partycjonowania. Na podstawie jego wartości dokonywany jest fizyczny podział. Obecnie istnieje możliwość tworzenia tabeli partycjonowanej bez klucza partycjonowania. Oznacza to, że nie zostały zdefiniowane żadne reguły, którymi SZBD mógłby się kierować podczas wyboru partycji. Partycje są statycznie definiowane przez administratora. Rzadko występuje przypadek braku jakiegokolwiek kryterium partycjonowania. Partycjonowanie systemowe pozwala na implementację zewnętrznego mechanizmu, który będzie decydował o wyborze partycji. Mechanizm ten powinien też wskazywać partycję, w której należy szukać danego rekordu.

W przypadku partycjonowania systemowego, również zastosowano uproszczoną procedurę zmiany struktury tabeli. Poniżej znajduje się struktura tabeli *TPOINTSTATE*, zawierająca dwie statycznie zdefiniowane partycje.

```
CREATE
  TABLE TPOINTSTATE
  (
    POINTSTATEID      NUMBER(38, 0),
    POINTSTATELEVELID NUMBER(1, 0),
    POINTSTATETYPEID  NUMBER(2, 0),
    ACCOUNTCID        NUMBER(15,0),
    ALLIANCECID       NUMBER(5,0),
    CHECKPOINTDATE    DATE,
    POINTS            NUMBER(10, 0)
  )
  PARTITION BY SYSTEM
  (
    PARTITION PPOINTSTATE1,
    PARTITION PPOINTSTATE2
  );
```

W instrukcji *DML* należy jawnie zadeklarować, do której partycji ma zostać zapisy rekord. W przeciwnym przypadku wystąpi błąd „*ORA-14701: partition-extended name or bind variable must be used for DMLs on tables partitioned by the System method*”.

```
INSERT
INTO
  TPOINTSTATER PARTITION
  (
    PPOINTSTATE1
  )
  VALUES
  (
    10000000,
    1,
    1,
```

```
    5623,
    1049,
    TO_DATE('01-01-13 00:00:00'),
    999
);
```

Również podczas instrukcji *SELECT* warto wskazać partycję, w której należy szukać rekordu. Można tego dokonać za pomocą klauzuli *PARTITION*. Jeśli nie zostanie to zrobione, partycjonowanie straci swój najważniejszy atut, ponieważ cała tabela będzie przeszukiwana zamiast pojedynczej partycji.

```
SELECT
  *
FROM
  TPOINTSTATER PARTITION (PPOINTSTATE1);
```

4.6. Wyzwalacze

Logika aplikacji, w obecnym stadium zaawansowania, nie opiera się w żaden sposób na wyzwalaczach. Bezpośrednie wykonywanie instrukcji *INSERT*, *UPDATE* i *DELETE* dla tabel jest niepożądane. Z tego też względu, przedstawione przykłady nie są wykorzystane do realizacji projektu oraz nie spowodowały, że stał się on lepszy dzięki nowej funkcjonalności. Ograniczają się one do prezentacji składni oraz działania, poprzez możliwość zaobserwowania sekwencji ciągów znaków pojawiających się w buforze wyjściowym.

4.6.1. Wyzwalacze złożone

Dotychczas, jeden wyzwalacz nie mógł wykonać fragmentu kodu przed i po zdarzeniu. Możliwe było uruchomienie wyzwalacza dla tego samemu momentu i różnych zdarzeń. Ponadto, wyzwalacz mógł być zdefiniowany jednocześnie tylko na poziomie instrukcji lub wiersza. Powyższe ograniczenia są niwelowane przez złożone wyzwalacze, które umożliwiają realizację następujących scenariuszy [5]:

- Blok kodu, który jest uruchamiany przed i po jednym zdarzeniu.
- Blok kodu, który jest uruchamiany przed i po wielu zdarzeniach.
- Blok kodu, który jest uruchamiany przed i po wielu zdarzeniach, na poziomie instrukcji i wiersza.

Poniżej znajduje się przykładowy wyzwalacz złożony, który wykonuje określone bloki kodu przed i po wystąpieniu instrukcji *UPDATE* dla tabeli *TACCOUNT*. Jest to jego najprostsza możliwa wersja. Należy zwrócić uwagę na globalną cześć, która jest dostępna

dla wszystkich bloków kodu w wyzwalaczu. Znajdują się w niej definicje dwóch zmiennych znakowych, które są wyświetlane w lokalnych blokach.

```
CREATE OR REPLACE TRIGGER TEST1 FOR UPDATE ON TACCOUNT COMPOUND
TRIGGER
  VVBEFOREUPST VARCHAR2(50) := 'PRZED UPDATE STATEMENT';
  VVAFTERUPST VARCHAR2(50) := 'AFTER UPDATE STATEMENT';
  BEFORE STATEMENT
IS
BEGIN
  DBMS_OUTPUT.PUT_LINE(VVBEFOREUPST);
END BEFORE STATEMENT;
AFTER STATEMENT
IS
BEGIN
  DBMS_OUTPUT.PUT_LINE(VVAFTERUPST);
END AFTER STATEMENT;
END;
```

Drugi możliwy przypadek to wyzwalacz złożony, który wykonuje określone bloki kodu przed i po wystąpieniu instrukcji *INSERT*, *UPDATE* lub *DELETE*.

```
CREATE OR REPLACE TRIGGER TEST2 FOR INSERT OR
  UPDATE OR
  DELETE
    ON TACCOUNT COMPOUND TRIGGER
      VVBEFOREUPST  VARCHAR2(50) := 'BEFORE UPDATE STATEMENT';
      VVAFTERUPST   VARCHAR2(50) := 'AFTER UPDATE STATEMENT';
      VVBEFOREINST  VARCHAR2(50) := 'BEFORE INSERT STATEMENT';
      VVAFTERINST   VARCHAR2(50) := 'AFTER INSERT STATEMENT';
      VVBEFOREDEST  VARCHAR2(50) := 'BEFORE DELETE STATEMENT';
      VVAFTERDEST   VARCHAR2(50) := 'AFTER DELETE STATEMENT';
BEFORE STATEMENT
IS
BEGIN
  CASE
  WHEN INSERTING THEN
    DBMS_OUTPUT.PUT_LINE(VVBEFOREINST);
  WHEN UPDATING THEN
    DBMS_OUTPUT.PUT_LINE(VVBEFOREUPST);
  WHEN DELETING THEN
    DBMS_OUTPUT.PUT_LINE(VVBEFOREDEST);
  END CASE;
END BEFORE STATEMENT;
AFTER STATEMENT
IS
BEGIN
  CASE
  WHEN INSERTING THEN
    DBMS_OUTPUT.PUT_LINE(VVAFTERINST);
  WHEN UPDATING THEN
    DBMS_OUTPUT.PUT_LINE(VVAFTERUPST);
  WHEN DELETING THEN
    DBMS_OUTPUT.PUT_LINE(VVAFTERDEST);
  END CASE;
END AFTER STATEMENT;
END;
```

Najbardziej rozbudowana wersja wyzwalacza złożonego pozwala obsłużyć kompleksowo wszystkie możliwe kombinacje na poziomie instrukcji i wiersza. Wyzwalacz wykonuje określone bloki kodu przed i po wystąpieniu instrukcji *INSERT*, *UPDATE* lub *DELETE* a także dla pojedynczych wierszy, na które wpływają te instrukcje.

```
CREATE OR REPLACE TRIGGER TEST3 FOR INSERT OR
  UPDATE OR
  DELETE
    ON TACCOUNT COMPOUND TRIGGER
VVBEFOREUPST VARCHAR2(50) := 'BEFORE UPDATE STATEMENT';
VVAFTERUPST  VARCHAR2(50) := 'AFTER UPDATE STATEMENT';
VVBEFOREINST VARCHAR2(50) := 'BEFORE INSERT STATEMENT';
VVAFTERINST  VARCHAR2(50) := 'AFTER INSERT STATEMENT';
VVBEFOREDEST VARCHAR2(50) := 'BEFORE DELETE STATEMENT';
VVAFTERDEST  VARCHAR2(50) := 'AFTER DELETE STATEMENT';
VVBEFOREUPR  VARCHAR2(50) := 'BEFORE UPDATE ROW';
VVAFTERUPR   VARCHAR2(50) := 'AFTER UPDATE ROW';
VVBEFOREINR  VARCHAR2(50) := 'BEFORE INSERT ROW';
VVAFTERINR   VARCHAR2(50) := 'AFTER INSERT ROW';
VVBEFOREDER  VARCHAR2(50) := 'BEFORE DELETE ROW';
VVAFTERDER   VARCHAR2(50) := 'AFTER DELETE ROW';
  BEFORE STATEMENT
IS
BEGIN
  CASE
  WHEN INSERTING THEN
    DBMS_OUTPUT.PUT_LINE(VVBEFOREINST);
  WHEN UPDATING THEN
    DBMS_OUTPUT.PUT_LINE(VVBEFOREUPST);
  WHEN DELETING THEN
    DBMS_OUTPUT.PUT_LINE(VVBEFOREDEST);
  END CASE;
END BEFORE STATEMENT;
AFTER STATEMENT
IS
BEGIN
  CASE
  WHEN INSERTING THEN
    DBMS_OUTPUT.PUT_LINE(VVAFTERINST);
  WHEN UPDATING THEN
    DBMS_OUTPUT.PUT_LINE(VVAFTERUPST);
  WHEN DELETING THEN
    DBMS_OUTPUT.PUT_LINE(VVAFTERDEST);
  END CASE;
END AFTER STATEMENT;
BEFORE EACH ROW
IS
BEGIN
  CASE
  WHEN INSERTING THEN
    DBMS_OUTPUT.PUT_LINE(VVBEFOREINR);
  WHEN UPDATING THEN
    DBMS_OUTPUT.PUT_LINE(VVBEFOREUPR);
  WHEN DELETING THEN
    DBMS_OUTPUT.PUT_LINE(VVBEFOREDER);
  END CASE;
END BEFORE EACH ROW;
```

```
AFTER EACH ROW
IS
BEGIN
  CASE
  WHEN INSERTING THEN
    DBMS_OUTPUT.PUT_LINE(VVAFTERINR);
  WHEN UPDATING THEN
    DBMS_OUTPUT.PUT_LINE(VVAFTERUPR);
  WHEN DELETING THEN
    DBMS_OUTPUT.PUT_LINE(VVAFTERDER);
  END CASE;
END AFTER EACH ROW;
END;
```

4.6.2. Sterowanie porządkiem wykonania

W przypadku występowania wielu wyzwalaczy, uruchamianych w efekcie dokładnie tego samego zdarzenia, momentu i poziomu, kolejność wykonania była przypadkowa. Rozwiązaniem tego problemu było tworzenie tylko i wyłącznie jednego wyzwalacza, dla każdego zdarzenia, momentu uruchomienia i poziomu, zawierającego niezwiązane ze sobą logicznie fragmenty kodu. Obejście problemu powodowało prawidłowe zachowanie systemu, ponieważ porządek wykonania był jawnie narzucony poprzez kolejność linii w kodzie źródłowym. Skutkiem ubocznym było trudniejsze utrzymanie oraz zbyt duża odpowiedzialność ciążąca na jednym obiekcie słownikowym.

Obecnie można tworzyć dowolną ilość wyzwalaczy uruchamianych przy tym samym zdarzeniu, momencie i poziomie, zapewniając odpowiednią kolejność wykonania. Służy do tego klauzula *FOLLOWS*. Dzięki nowej funkcjonalności zachowana jest modułowość [7]. Fragmenty kodu źródłowego, odpowiedzialne za poszczególne etapy przetwarzania danych, mogą znajdować się w osobnych wyzwalaczach.

Utworzono dwa bardzo proste wyzwalacze, narzucając jasno porządek wykonania. Zgodnie z oczekiwaniami, wykonały się one w prawidłowej kolejności.

```
CREATE OR REPLACE TRIGGER PIERWSZY BEFORE
  INSERT
    ON TACCOUNT BEGIN DBMS_OUTPUT.PUT_LINE('PIERWSZY');
END;

CREATE OR REPLACE TRIGGER DRUGI BEFORE
  INSERT
    ON TACCOUNT FOLLOWS PIERWSZY BEGIN DBMS_OUTPUT.PUT_LINE('DRUGI');
END;
```

Przetestowano zachowanie tych samych wyzwalaczy z pominięciem klauzuli *FOLLOWS*. W takim przypadku, wyzwalacz drugi wykonywał się zawsze przed

wyzwalaczem pierwszym. Kolejność kompilacji nie miała wpływu na porządek wykonania.

4.7. Kompresja tabeli

Kompresja tabeli pozwala na zmniejszenie zajmowanego przez nią miejsca w przestrzeni tabel. Funkcjonalność została wprowadzona wraz z *Oracle Database 9i* ale była dedykowana głównie dla hurtowni danych i dostępna tylko dla masowych operacji wstawiania (*Direct-Path INSERT*), które pomijają całkowicie bufor instancji zapisując dane bezpośrednio do plików bazy danych. Została ona ulepszona wraz z nową wersją. Obecnie, mechanizm może być swobodnie wykorzystywany w systemach transakcyjnych [12]. Jest dostępny dla wszystkich instrukcji *DML* i może być stosowany w pojedynczych i masowych operacjach.

Głównym zamierzeniem nie jest oszczędność przestrzeni dyskowej, ale zwiększenie wydajności operacji *I/O*. Może to wpływać pozytywnie na czas wykonania zapytań, zwłaszcza przy pełnych odczytach [12]. Zwiększenie przepustowości wiąże się z większym zapotrzebowaniem czasu procesora na kompresję i dekompresję, ponieważ operacje są wykonywane w locie. Kompresja jest w pełni przeźroczysta dla aplikacji korzystających z bazy danych. Może być wykorzystywana w kontekście tabeli lub też pojedynczych partycji. Przestrzeń tabel potrafi zdefiniować, że wszystkie tabele tworzone w niej mają być domyślnie kompresowane, jednak podczas tworzenia istnieje możliwość przesłonięcia tego ustawienia i sprawienia, że tabela nie będzie przechowywana w skompresowanej formie. W tabeli 28 umieszczone wszystkie możliwe opcje kompresji.

Klauzula	**Znaczenie**
NOCOMPRESS	Oznacza brak kompresji. Jeśli nie zdefiniowano innej domyślnej wartości przy tworzeniu przestrzeni tabel, to właśnie ta opcja jest domyślna.
COMPRESS FOR DIRECT_LOAD OPERATIONS	Kompresja włączona tylko dla masowych operacji ładowania danych. Działa ona dokładnie tak samo jak w starszych wersjach *Oracle Database*.

Klauzula	Znaczenie
COMPRESS	Alias dla opcji *COMPRESS FOR DIRECT_LOAD OPERATIONS*. Obie oznaczają dokładnie to samo.
COMPRESS FOR ALL OPERATIONS	Kompresja włączona dla wszystkich operacji *DML*.

Tabela 28. Dostępne opcje kompresji tabel.

Zostały utworzone dwie tabele testowe, *TPOINTSTATECOMPRESSED* i *TPOINTSTATECOMPRESSED* o identycznej strukturze jak tabela *TPOINTSTATE*.

```
CREATE TABLE TPOINTSTATECOMPRESSED
  (
    POINTSTATEID      NUMBER(38, 0),
    POINTSTATELEVELID NUMBER(1, 0),
    POINTSTATETYPEID  NUMBER(2, 0),
    ACCOUNTCID        NUMBER(15,0),
    ALLIANCECID       NUMBER(5,0),
    CHECKPOINTDATE    DATE,
    POINTS            NUMBER(10, 0),
    CONSTRAINT "CPKTPOINTSTATEIDT" PRIMARY KEY ("POINTSTATEID")

  )
COMPRESS FOR ALL OPERATIONS;

CREATE TABLE TPOINTSTATEUNCOMPRESSED
  (
    POINTSTATEID      NUMBER(38, 0),
    POINTSTATELEVELID NUMBER(1, 0),
    POINTSTATETYPEID  NUMBER(2, 0),
    ACCOUNTCID        NUMBER(15,0),
    ALLIANCECID       NUMBER(5,0),
    CHECKPOINTDATE    DATE,
    POINTS            NUMBER(10, 0),
    CONSTRAINT "CPKTPOINTSTATEIDT2" PRIMARY KEY ("POINTSTATEID")

  )
```

Wstawiono do nich wszystkie wiersze znajdujące się w tabeli *TPOINTSTATE*. Pozwoli to wyznaczyć współczynnik kompresji.

```
INSERT
INTO
  TPOINTSTATECOMPRESSED
SELECT
  *
FROM
  TPOINTSTATE;

INSERT
INTO
  TPOINTSTATEUNCOMPRESSED
SELECT
  *
FROM
  TPOINTSTATE;
```

Z widoku *USER_EXTENTS* można uzyskać informację o przestrzeni dyskowej, jaką zajmują wszystkie zaalokowane ekstenty. Z tabeli 29 wynika, że stopień kompresji wynosi 0,75.

```
SELECT
  SEGMENT_NAME,
  SUM(BYTES) / 1024 / 1024 AS SIZE_MB
FROM
  USER_EXTENTS
WHERE
  SEGMENT_NAME IN
('TPOINTSTATEUNCOMPRESSED',
'TPOINTSTATECOMPRESSED')
GROUP BY
  SEGMENT_NAME;
```

SEGMENT_NAME	SIZE_MB
TPOINTSTATECOMPRESSED	6
TPOINTSTATEUNCOMPRESSED	8

Tabela 29. Obszar zaalokowany (E) dla tabeli skompresowanej i nieskompresowanej.

Nie jest to jednak najlepszy sposób na sprawdzenie rozmiaru. Tabela może posiadać prawie pusty ekstent 128 blokowy o wielkości 1 MB, który dopiero został zaalokowany. Innym, bardziej dokładnym, jest zgromadzenie statystyk oraz odczytanie ilości zajętych bloków. Każdy blok ma standardowy rozmiar 8 KB.

```
EXECUTE DBMS_STATS.GATHER_TABLE_STATS('OIMS',
'TPOINTSTATECOMPRESSED');
EXECUTE DBMS_STATS.GATHER_TABLE_STATS('OIMS',
'TPOINTSTATEUNCOMPRESSED');
```

Nowy współczynnik kompresji 0,748, wyznaczony na podstawie tabeli 30, nie odbiega znacząco od pierwotnej liczby. Skompresowana tabela zawiera tylko i wyłącznie dane liczbowe. W przypadku obiektu z polami tekstowymi można się spodziewać lepszego stopnia kompresji.

```
SELECT
  TABLE_NAME,BLOCKS,
  (BLOCKS * 8 / 1024) AS SIZE_MB
FROM
  DBA_TAB_PENDING_STATS
WHERE
  OWNER          = 'OIMS'
AND TABLE_NAME IN ('TPOINTSTATEUNCOMPRESSED',
'TPOINTSTATECOMPRESSED');
```

TABLE_NAME	BLOCKS	SIZE_MB
TPOINTSTATECOMPRESSED	748	5.84375
TPOINTSTATEUNCOMPRESSED	1000	7.8125

Tabela 30. Obszar zaalokowany (B) dla tabeli skompresowanej i nieskompresowanej.

W celu zbadania wydajności instrukcji *DML*, został przeprowadzony test polegający na wykonaniu dla obu tabel 100 tysięcy razy poleceń *INSERT*, *UPDATE* oraz *DELETE*. Symuluje on zachowanie systemu transakcyjnego, w którym wykonywane są duże ilości niewielkich operacji. Z pomocą przyszło narzędzie *SQL TRACE*, które pozwala na zapis do pliku szczegółowych informacji o wykonywanych instrukcjach w danej sesji. Korzystając z niego można uzyskać [18]:

- Całkowity czas wykonania,
- Wykorzystany czas procesora,
- Ilość odczytanych bloków,
- Ilość wykorzystanych jednostek bufora;

Na pracę narzędzia wpływają parametry instancji opisane w tabeli 31. Domyślne wartości sprawiają, że wszystkie niezbędne informacje zostaną uzyskane.

Nazwa parametru	**Opis**	**Bieżąca wartość**
TIMED_STATISTICS	Odpowiada za sterowanie gromadzeniem informacji o całkowitym czasie wykonania i wykorzystanym czasie procesora.	TRUE
MAX_DUMP_FILE_SIZE	Określa maksymalną wielkość pojedynczego pliku śladu. Wartość jest określona w blokach systemu plików.	UNLIMITED
USER_DUMP_DEST	Określa ścieżkę zapisu plików śladu.	/u01/app/oracle/diag/rdbms/testdbds/testdbds/trace

Tabela 31. Parametry instancji bazy danych wpływające na narzędzie *SQL TRACE*.

Po rozpoczęciu sesji, włączono śledzenie poleceń *SQL*, wykonano masowe wstawianie, aktualizację oraz usuwanie a następnie wyłączono śledzenie.

```
ALTER SESSION SET SQL_TRACE TRUE;

BEGIN
  FOR I IN 1..100000
  LOOP
    INSERT
    INTO
      TPOINTSTATECOMPRESSED VALUES
      (
        I,
        1,
        1,
        5623,
        1049,
        TO_DATE('01-01-13 00:00:00'),
```

```
        10000000
      );
  END LOOP;
END;

BEGIN
  FOR I IN 1..100000
  LOOP
    UPDATE
      TPOINTSTATECOMPRESSED
    SET
      POINTS = 10000
    WHERE
      POINTSTATEID = I;
  END LOOP;
END;

DECLARE
  REC TPOINTSTATECOMPRESSED%ROWTYPE;
BEGIN
  FOR I IN 1..100000
  LOOP
    SELECT
      *
    INTO
      REC
    FROM
      TPOINTSTATECOMPRESSED
    WHERE
      POINTSTATEID = I;
  END LOOP;
END;

BEGIN
  FOR I IN 1..100000
  LOOP
    DELETE
    FROM
      TPOINTSTATECOMPRESSED
    WHERE
      POINTSTATEID = I;
  END LOOP;
END;

ALTER SESSION SET SQL_TRACE FALSE;
```

Żądane informacje znajdują się w plikach o rozszerzeniu *trc*, które są nieczytelne. Użyto specjalnego narzędzia o nazwie *tkprof*. Parametr *explain* określa nazwę użytkownika oraz hasło do schematu, który ma zostać wykorzystany do przeprowadzenia analizy planu wykonania zapytań.

```
tkprof testdbds_ora_26411.trc compressed.ptr explain=system/manager
```

Nieczytelny plik został przekonwertowany do postaci tekstowej. Uzyskane wyniki są prezentowane w formie tabelarycznej z podziałem na trzy etapy przetwarzania instrukcji: kompilację, wykonanie i pobranie wyniku. W stopce znajduje się

podsumowanie. Najważniejsze parametry wyniku pomiaru wydajności umieszczono w tabeli 32.

Tabela	Koszt pełnego odczytu	Instrukcja	Czas wykonania [s]	Czas procesora [s]	Liczba wykorzystanych buforów
Nieskompresowana	274	INSERT	6,13	6,08	306632
		UPDATE	6,04	5,97	102222
		DELETE	7,27	7	308124
		SELECT	2,58	2,51	300000
Skompresowana	206	INSERT	6,61	6,71	311778
		UPDATE	9,02	9,08	593642
		DELETE	7,9	7,71	397803
		SELECT	2,67	2,55	369176

Tabela 32. Porównanie wydajności operacji dla tabeli z kompresją i bez kompresji.

Zgodnie z oczekiwaniami, mniejszy rozmiar tabeli skutkuje niższym kosztem pełnego dostępu. Zwiększenie przepustowości operacji *I/O* zostało osiągnięte kosztem wyższego wykorzystania czasu procesora. Całkowity czas wykonania również wzrósł. Różnice nie są jednak na tyle duże, aby zdyskwalifikować mechanizm i sklasyfikować go jako nieprzydatny.

4.8. Technologia *Flashback*

Funkcjonalność *Flashback* jest zbiorem narzędzi, które pozwalają na oglądanie stanu bazy danych lub obiektów bazy danych z przeszłości oraz powrotu do ich poprzednich stanów bez przestoju instancji bazy danych [19]. Technologia została zapoczątkowana bardzo skromnie w wersji *Oracle Database 9i*, poprzez dodanie funkcji *Flashback Query*, która umożliwia podgląd tabeli w stanie z przeszłości poprzez dodanie klauzuli *AS OF* w instrukcji *SELECT*. Korzystając z *Flashback* operuje się znacznikami czasowymi, numerami *SCN* (*System Change Number*) lub też punktami przywracania. W *Oracle Database 10g* znacząco rozbudowano *Flashback* poprzez dodanie następujących funkcji:

- *Flashback Database* – przywracanie całej bazy danych do stanu ze znacznika czasowego, numeru *SCN* lub punktu przywracania,
- *Flashback Drop* – odzyskiwanie usuniętej tabeli z kosza,

- *Flashback Table* – przywracanie całej tabeli do stanu ze znacznika czasowego, numeru SCN lub punktu przywracania,
- *Flashback Version Query* – przeglądanie wszystkich wersji zdefiniowanych wierszy w danym okresie,
- *Flashback Transaction Query* – przeglądanie zmian dokonanych na poziomie transakcji;

W *Oracle Database 11g* dodano kolejne zaawansowane mechanizmy wchodzące w skład grupy *Flashback*. Są to:

- *Flashback Transaction* – wycofanie dowolnej zatwierdzonej transakcji wraz zależnościami,
- *Flashback Data Archive* – śledzenie i przechowywanie wszystkich zmian zachodzących w wybranych tabelach;

4.8.1. Dostosowanie instancji bazy danych

Na działanie funkcjonalności Flashback wpływają dwa parametry systemowe. Pierwszym z nich jest *UNDO_RETENTION*, który definiuje w sekundach, jak daleko będzie sięgać zawartość przestrzeni wycofania. Domyślną wartością jest 900 [13], co teoretycznie pozwala na oglądanie tabel za pomocą klauzuli *AS OF* maksymalnie sprzed jednego kwadransa. Niestety, wartość jest tylko wskazówką dla instancji bazy danych, aby jedynie starała się utrzymać zdefiniowany okres retencji. To przestrzeń tabel określa czy czas retencji jest gwarantowany czy nie. Można to zweryfikować poprzez poniższe zapytanie.

```
SELECT RETENTION FROM DBA_TABLESPACES WHERE CONTENTS = 'UNDO';
```

Wartość kolumny wynosi „*NOGUARANTEE*”, co oznacza, że okres retencji nie jest gwarantowany. Aby mieć pewność, że w każdej chwili istnieje możliwość obejrzenia tabeli sprzed kwadransa, należy zmienić przestrzeń tabel.

```
ALTER TABLESPACE UNDOTBS1 RETENTION GUARANTEE;
```

Drugim parametrem jest *DB_FLASHBACK_RETENTION_TARGET*, który definiuje w minutach jak daleko w czasie sięgają informacje pozwalające na przywrócenie całej bazy danych. Domyślną wartością jest 1440 [13], co pozwala na przywrócenie bazy danych maksymalnie do stanu sprzed 24 godzin.

Aby w pełni korzystać z technologii *Flashback*, instancja bazy danych musi pracować w trybie archiwizacji, który permanentnie zapisuje logi dziennika powtórzeń. Domyślnie, archiwizacja jest wyłączona. Aby zmienić tryb pracy bazy danych należy ją wyłączyć i uruchomić ponownie, jako zamontowaną, ale nieotwartą.

```
SHUTDOWN IMMEDIATE;
STARTUP MOUNT;
```

Na zamontowanej instancji bazy danych zmieniono tryb pracy na *ARCHIVELOG*, wymuszono zmianę bieżącej grupy plików dziennika powtórzeń, włączono funkcję *Flashback Database* oraz otworzono instancję bazy danych.

```
ALTER DATABASE ARCHIVELOG;
ALTER SYSTEM ARCHIVE LOG CURRENT;
ALTER DATABASE FLASHBACK ON;
ALTER DATABASE OPEN;
```

Aby Oracle *Flashback Transaction Query* oraz *Flashback Transaction* działały, włączono minimalne logowanie uzupełniające oraz logowanie kluczy głównych tabel dla dziennika powtórzeń. Pozwoli to na rozwiązywanie zależności pomiędzy transakcjami wykonanymi w bazie danych.

```
ALTER DATABASE ADD SUPPLEMENTAL LOG DATA;
ALTER DATABASE ADD SUPPLEMENTAL LOG DATA (PRIMARY KEY) COLUMNS;
```

Istnieje też możliwość logowania uzupełniającego kluczy obcych jednak liczba takowych ograniczeń jest z reguły dużo większa niż kluczy głównych. Może to mieć negatywny wpływ na wydajność systemu [19]. Bez tej opcji system nie wykrywa zależności klucza obcego.

```
ALTER DATABASE ADD SUPPLEMENTAL LOG DATA (FOREIGN KEY) COLUMNS;
```

Aby sprawdzić czy mechanizmy *Flashback*, bazujące na przestrzeni wycofania, funkcjonują, wykonano aktualizację dla jednego wiersza w tabeli.

```
UPDATE TACCOUNT SET USERNAME = 'Test' WHERE USERNAME = 'Destro';
```

Mechanizm *Flashback* powinien gwarantować możliwość obejrzenia zawartości tabeli w postaci, w jakiej była 15 minut temu. Skorzystano z *Flashback Query*, aby wyświetlić obraz tabeli z teoretycznie najstarszej możliwej daty.

```
SELECT * FROM TACCOUNT AS OF TIMESTAMP (SYSTIMESTAMP - INTERVAL '15'
MINUTE) WHERE USERNAME = 'Destro';
```

Rezultatem jest wiersz niezmodyfikowany. To samo zapytanie, ale z pominiętą klauzulą *AS OF* zwróciłoby nową wersję wiersza. Dla wykorzystanej tabeli, udało się zobaczyć obraz nawet sprzed 3000 minut, chociaż tylko 15 jest gwarantowane. Powodem

jest to, że niewiele operacji zostało wykonanych na bazie danych od tamtej pory i przestrzeń wycofania zawiera nadal bardzo stare informacje. Jeśli nie ma możliwości objerzenia zawartości z danego znacznika czasu to występuje błąd „*ORA-08180: no snapshot found based on specified time*”.

4.8.2. Flashback Database

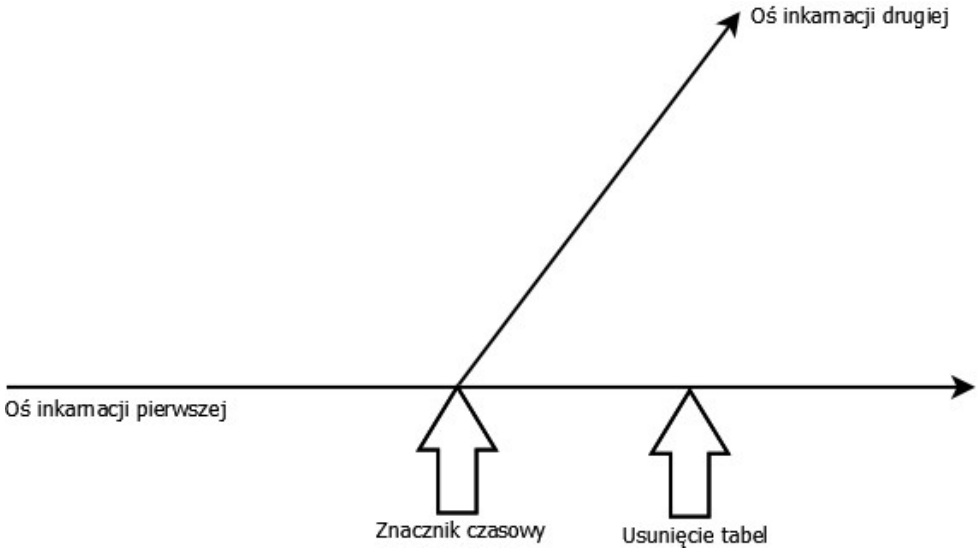
Funkcja pozwala na przywrócenie całej bazy danych do znacznika czasowego, numeru *SCN* lub punktu przywracania [20]. Konieczne są uprawnienia *SYSDBA*, aby skorzystać z tejże funkcjonalności. Aby przetestować działanie, wszystkie tabele schematu *OIMS* wraz z ograniczeniami zostały usunięte tak, aby nie można było ich przywrócić z kosza.

```
DROP TABLE TLOADEDCHECKPOINT CASCADE CONSTRAINTS PURGE;
DROP TABLE TPERIOD CASCADE CONSTRAINTS PURGE;
DROP TABLE TDIFFERENCE CASCADE CONSTRAINTS PURGE;
DROP TABLE TPOINTSTATE CASCADE CONSTRAINTS PURGE;
DROP TABLE TPOINTSTATELEVEL CASCADE CONSTRAINTS PURGE;
DROP TABLE TPOINTSTATETYPE CASCADE CONSTRAINTS PURGE;
DROP TABLE TPLANET CASCADE CONSTRAINTS PURGE;
DROP TABLE TACCOUNT CASCADE CONSTRAINTS PURGE;
DROP TABLE TALLIANCE CASCADE CONSTRAINTS PURGE;
DROP TABLE TUNIVERSE CASCADE CONSTRAINTS PURGE;
DROP TABLE TSERVER CASCADE CONSTRAINTS PURGE;
DROP TABLE TUNIVERSENAME CASCADE CONSTRAINTS PURGE;
DROP TABLE TGAMEAREA CASCADE CONSTRAINTS PURGE;
```

W celu przywrócenia bazy danych to stanu sprzed jednej godziny należy wyłączyć instancję, wykonać instrukcję *FLASHBACK DATABASE TO TIMESTAMP* oraz otworzyć bazę danych z opcją *RESETLOGS*, która tworzą nową inkarnację instancji bazy danych.

```
SHUTDOWN IMMEDIATE;
STARTUP MOUNT;
FLASHBACK DATABASE TO TIMESTAMP SYSTIMESTAMP-1/24;
ALTER DATABASE OPEN RESETLOGS;
```

Nowa inkarnacja bazy danych jest tworzona przy każdym przywróceniu. Historia bazy danych rozgałęzia się, co widać na rysunku 8. Po momencie usunięcia tabel nastąpiło przywrócenie do znacznika czasowego.



Rysunek 8. Wizualizacja inkarnacji bazy danych po przywróceniu.

Istnieje możliwość przywrócenia bazy danych w przyszłość, do punktu znajdującego się za znacznikiem czasowym, czyli po chwili, w której nastąpiło rozgałęzienie. Są stworzone do tego specjalne narzędzia, które na podstawie zarchiwizowanych plików dziennika powtórzeń odtwarzają stan bazy danych do momentu awarii. Najwygodniej jednak zasymulować taki scenariusz za pomocą punktów przywracania. Utworzono pierwszy punkt przywracania.

```
CREATE RESTORE POINT PIERWSZY GUARANTEE FLASHBACK DATABASE;
```

Następnie ponownie usunięto wszystkie tabele ze schematu *OIMS* oraz utworzono drugi punkt przywracania.

```
CREATE RESTORE POINT DRUGI GUARANTEE FLASHBACK DATABASE;
```

Po przywróceniu bazy danych do pierwszego punktu przywracania, wszystkie tabele istniały. Po przywróceniu bazy danych do drugiego punktu przywracania, z przyszłości, tabele nie istniały. Mechanizm zachował się tak jak oczekiwano. Po wykonaniu testu, ponownie przywrócono bazę danych do należytego stanu tzn. do momentu, kiedy nie ma żadnego uszczerbku. Za opisane czynności odpowiada poniższa sekwencja instrukcji.

```
SHUTDOWN IMMEDIATE;
STARTUP MOUNT;
FLASHBACK DATABASE TO RESTORE POINT PIERWSZY;
ALTER DATABASE OPEN RESETLOGS;
SHUTDOWN IMMEDIATE;
STARTUP MOUNT;
FLASHBACK DATABASE TO RESTORE POINT DRUGI;
ALTER DATABASE OPEN RESETLOGS;
SHUTDOWN IMMEDIATE;
```

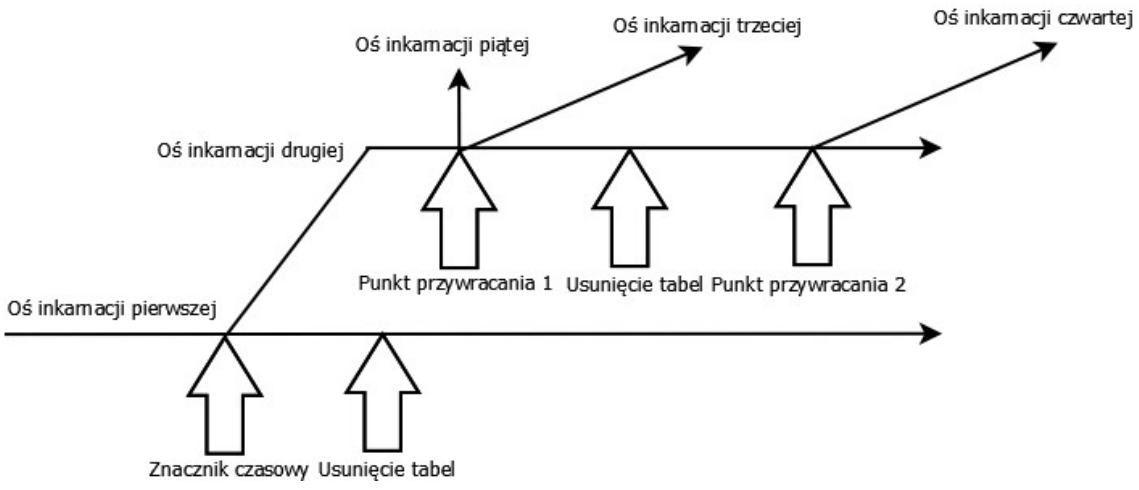
```
STARTUP MOUNT;
FLASHBACK DATABASE TO RESTORE POINT PIERWSZY;
ALTER DATABASE OPEN RESETLOGS;
```

Istnieje możliwość śledzenia inkarnacji bazy danych poprzez widok *V$DATABASE_INCARNATION*. Zawiera on m.in. informacje przedstawione w tabeli 33. Dwie pierwsze kolumny zawierają numer początkowy *SCN* danej inkarnacji oraz czas utworzenia. Status, oprócz wskazania bieżącej inkarnacji, pokazuje, które z nich są rodzinami lub zostały porzucone. W rozważanym przypadku, który został przedstawiony na rysunku 9, dwie inkarnacje są rodzicami (pierwsza oraz druga) oraz dwie zostały porzucone (trzecia oraz czwarta). Ostatnia flaga informuje czy możliwe jest skorzystanie z *Flashback Database* w celu przywrócenia bazy danych do jakiegoś punktu w danej inkarnacji.

```
SELECT
  RESETLOGS_CHANGE#,
  RESETLOGS_TIME,
  STATUS,
  FLASHBACK_DATABASE_ALLOWED
FROM
  V$DATABASE_INCARNATION;
```

RESETLOGS_CHANGE#	RESETLOGS_TIME	STATUS	FLASHBACK_DATABASE_ALLOWED
945184	04-01-13	PARENT	YES
12291129	04-01-13	PARENT	YES
12295016	04-01-13	ORPHAN	YES
12295982	04-01-13	ORPHAN	YES
12295016	04-01-13	CURRENT	YES

Tabela 33. Wybrane parametry inkarnacji bazy danych.



Rysunek 9. Wizualizacja pełnej historii inkarnacji bazy danych.

4.8.3. Flashback Drop

Funkcjonalność opisywana w tymże podrozdziale jest chyba najprostszą z całej rodziny. *Flashback Drop* pozwala na przywrócenie obiektu do stanu sprzed usunięcia, o ile znajduje się on w koszu [19]. W przeciwnym przypadku wystąpi błąd „*ORA-38305: object not in RECYCLE BIN*". Usunięto tabelę *TDIFFERENCE*.

```
DROP TABLE TDIFFERENCE;
```

Aby chwilę po błędzie przywrócić ją, wydano poniższe polecenie. Obiekt został przywrócony z dokładnie taką sama nazwą jak oryginał.

```
FLASHBACK TABLE TDIFFERENCE TO BEFORE DROP;
```

Jeśli w koszu znajduje się wiele obiektów, które oryginalną nazwę mają taką samą jak przywracana tabela to zostanie wybrany rekord, który trafił do kosza najpóźniej. Taka sytuacja jest zaprezentowana w tabeli 34.

```
SELECT
  OBJECT_NAME,
  ORIGINAL_NAME
FROM
  USER_RECYCLEBIN;
```

OBJECT_NAME	ORIGINAL_NAME
BIN$0oCbLU35+RXgQAB/AQB/0A==$0	TDIFFERENCE
BIN$0oCbLU34+RXgQAB/AQB/0A==$0	TDIFFERENCE

Tabela 34. Zawartość kosza po dwukrotnym usunięciu tabeli o tej samej nazwie.

Aby przywrócić kopię inną niż ostatnia, skorzystano z instrukcji rozbudowanej o klauzulę *RENAME TO*. Wykorzystuje ona nazwę obiektu z kosza oraz wskazuje nazwę docelową.

```
FLASHBACK TABLE "BIN$0oCbLU34+RXgQAB/AQB/0A==$0" TO BEFORE DROP RENAME
TO TDIFFERENCE;
```

4.8.4. Flashback Table

Flashback Table pozwala na przywrócenie tabeli, do stanu ze znacznika czasowego, numeru *SCN* lub punktu przywracania. Opcja jest dedykowana, jako rozwiązanie w sytuacji, kiedy popełniono błędy przy modyfikacji danych oraz zatwierdzono transakcję. Aby wykonanie powiodło się, tabela musi mieć włączoną opcję przenoszenia wierszy.

```
ALTER TABLE TACCOUNT ENABLE ROW MOVEMENT;
```

Typowym błędem, popełnianym podczas modyfikacji danych, jest brak warunku. Wykonano instrukcję *UPDATE* oraz zatwierdzono transakcję.

```
UPDATE TACCOUNT SET USERNAME = 'Test';
COMMIT;
```

Pomyłkę można szybko naprawić poprzez przywrócenie stanu tabeli do takiego, w jakim była dokładnie dwie minuty temu. Po wykonaniu instrukcji, tabela zawiera niezmodyfikowane rekordy.

```
FLASHBACK TABLE TACCOUNT TO TIMESTAMP SYSTIMESTAMP - INTERVAL '2'
MINUTE;
```

Warto zwrócić uwagę na klauzulę [*ENABLE*|*DISABLE*] *TRIGGERS*. Domyślnie, wyzwalacze nie są uruchamiane dla tabeli, która jest przywracana. Są one wyłączane przed rozpoczęciem operacji i włączane ponownie po zakończeniu. Wyzwalacze wyłączone przed uruchomieniem przywracania, pozostają wyłączone również po [10]. Wersja polecenia uwzględniająca działanie wyzwalaczy znajduje się poniżej.

```
FLASHBACK TABLE TACCOUNT TO TIMESTAMP SYSTIMESTAMP - INTERVAL '2'
MINUTE ENABLE TRIGGERS;
```

4.8.5. Flashback Version Query

Mechanizm *Flashback Version Query* umożliwia wyświetlenie wszystkich wersji danego wiersza, które występowały w zadanym okresie. Bazuje on na klauzuli *SELECT*. W wyniku istnieje możliwość skorzystania z kilku pseudokolumn, które zawierają informacje historyczne [19]. Zostały one wyliczone oraz opisane w tabeli 35.

Nazwa pseudokolumny	**Opis**
VERSIONS_STARTSCN	Numer SCN określający czas utworzenia danej wersji wiersza.
VERSIONS_STARTTIME	Znacznik czasowy utworzenia danej wersji wiersza.
VERSIONS_ENDSCN	Numer SCN określający czas wygaśnięcia danej wersji wiersza.
VERSIONS_ENDTIME	Znacznik czasowy wygaśnięcia danej wersji wiersza.
VERSIONS_XID	Identyfikator transakcji, która utworzyła daną wersję wiersza.

Nazwa pseudokolumny	Opis
VERSIONS_OPERATION	Operacja *DML* wykonana przez transakcję. Możliwymi wartościami są: ▪ I – instrukcja *INSERT*, wiersz przedstawia rekord po wykonaniu instrukcji, ▪ D – instrukcja *DELETE*, wiersz przedstawia rekord przed wykonaniem instrukcji, ▪ U – instrukcja *UPDATE*, wiersz przedstawia rekord po wykonaniu instrukcji;

Tabela 35. Pseudokolumny mechanizmu *Flashback Version Query*.

Zaktualizowano jeden wiersz *w tabeli TACCOUNT*, tak, aby istniały co najmniej dwie różne wersje jednego wiersza.

```
UPDATE
  TACCOUNT
SET
  USERNAME = 'Test'
WHERE
  USERNAME = 'Destro';
```

Przygotowano zapytanie, które wyświetlenia wszystkie wersje zmodyfikowanego wiersza pomiędzy znacznikiem czasu sprzed 15 minut oraz bieżącym czasem. Rezultat, przedstawiony w tabeli 36, pokazuje obie wersje wiersza tj. przed i po modyfikacji instrukcją *UPDATE*. Warto zwrócić szczególną uwagę na daty, w jakich obowiązywały poszczególne wersje wiersza.

```
SELECT
  USERNAME,
  VERSIONS_STARTTIME STARTTIME,
  VERSIONS_ENDTIME ENDTIME,
  VERSIONS_XID,
  VERSIONS_OPERATION OPERATION
FROM
  TACCOUNT VERSIONS BETWEEN TIMESTAMP SYSTIMESTAMP - 1/24/4 AND
SYSTIMESTAMP
WHERE
  ACCOUNTID = 1153
```

USERNAME	STARTIME	ENDTIME	VERSIONS_XID	OPERATION
Test	05-JAN-13 05.45.21.000000000 PM		06000000CF400000	U
Destro		05-JAN-13 05.45.21.000000000 PM		

Tabela 36. Rezultat zastosowania *Flashback Version Query* dla zmienionego wiersza.

4.8.1. Flashback Transaction Query

Flashback Transaction Query najczęściej wykorzystuje się w kooperacji z *Flashback Version Query*. Z funkcjonalnością związany jest widok *FLASHBACK_TRANSACTION_QUERY*. Zwraca on atomowe operacje *DML*, wykonywane w obrębie wybranej transakcji lub wszystkich transakcji w zadanym okresie. Dla każdej z nich jest wygenerowany kod *SQL*, który kompensuje zmianę. Można z niego skorzystać do wycofania tylko błędnego fragmentu całej transakcji. Zlokalizowano wszystkie operacje wykonane przez transakcję o identyfikatorze „06000000CF400000". Jest to transakcja użyta w rozdziale 4.8.5, który opisuje *Flashback Version Query*. W rezultacie, umieszczonym w tabeli 37, znajduje się instrukcja *UPDATE*, która zmieni wiersz do stanu sprzed wykonanej transakcji. Każdy rekord w widoku *FLASHBACK_TRANSACTION_QUERY*, odpowiada operacji wykonanej na dokładnie jednym wierszu.

```
SELECT
  START_SCN,
  COMMIT_SCN,
  OPERATION,
  TABLE_NAME,
  UNDO_SQL
FROM
  FLASHBACK_TRANSACTION_QUERY
WHERE
  XID = HEXTORAW('06000000CF400000');
```

START_SCN	COMMIT_SCN	OPERATION	TABLE_NAME	UNDO_SQL
12365948	12365967	UPDATE	TACCOUNT	update "OIMS"."TACCOUNT" set "USERNAME" = 'Destro' where ROWID = 'AAASl1AAEAAAAGGABK';

Tabela 37. Atomowe operacje wykonane w obrębie analizowanej transakcji.

4.8.2. Flashback Transaction

Flashback Transaction służy do wycofania dowolnej transakcji, na podstawie informacji zgromadzonych w widoku *FLASHBACK_TRANSACTION_QUERY* oraz dodatkowych informacji zapisywanych przez logowanie uzupełniające. Operacje kompensujące są wykonywane automatycznie, bez ingerencji administratora. Ponadto, mechanizm posiada bardziej istotną cechę, która polega na rozwiązaniu zależności

pomiędzy transakcjami [19]. Wykrywane scenariusze są przedstawione w tabeli 38. Zakłada się, że istnieją transakcje pierwsza i druga. Wycofanie transakcji pierwszej zależy od transakcji drugiej.

Zależność	**Opis**
Zapis po zapisie	Transakcja pierwsza modyfikuje wiersz. Następnie, transakcja druga zmieniła ten sam wiersz.
Klucz główny	W tabeli występuje wiersz z pewną wartością klucza głównego. Transakcja pierwsza usuwa ten rekord. Następnie, transakcja druga wstawia do rozważanej tabeli wiersz o tej samej wartości klucza głównego, co rekord usunięty przez transakcję pierwszą.
Klucz obcy	W tabeli występuje wiersz odnoszący się przez klucz obcy do klucza głównego innej tabeli. Transakcja pierwsza zmienia wartość klucza głównego (wymagane odroczone ograniczenie) a następnie transakcja druga zmienia wartość klucza obcego, który odnosił się do starej wartości klucza głównego.

Tabela 38. Zależności wykrywane przez *Flashback Transaction*.

Do wycofania transakcji służy procedura *TRANSACTION_BACKOUT* z pakietu *DBMS_FLASHBACK* [14]. Przyjmuje ona następujące parametry:

- Ilość transakcji do wycofania,
- Tablica z identyfikatorami transakcji do wycofania,
- Opcja definiująca zachowanie w stosunku do zależności;

Najważniejszym przekazywanym parametrem jest opcja, która determinuje zachowanie w stosunku do zależności. Możliwe wartości, wraz z opisami, zostały przedstawione w tabeli 39.

Wartość opcji	**Opis**
DBMS_FLASHBACK.NOCASCADE	Administrator zakłada, że zależności nie występują. Jeśli takowe istnieją to próba wycofania kończy się błędem.
DBMS_FLASHBACK.NOCASCADE_FORCE	Wszelkie zależności są ignorowane. Ten tryb jest najbardziej podobny do ręcznego wycofywania transakcji, ponieważ kompensujące operacje DML są wykonywane w odwrotnej kolejności. Po zakończeniu, administrator decyduje czy należy zatwierdzić wycofanie transakcji lub też anulować wycofanie.

Wartość opcji	Opis
DBMS_FLASHBACK.NONCONFLICT_ONLY	Wycofywane są tylko te operacje transakcji, które nie powodują występowania zależności. Atomowość transakcji może zostać utracona w tym przypadku. Podobny efekt zostanie osiągnięty poprzez ręczne uruchomienie wybranych instrukcji kompensujących z widoku *FLASHBACK_TRANSACTION_QUERY*.
DBMS_FLASHBACK.CASCADE	Wycofanie transakcji wraz z ze wszystkimi zależnościami. Przechodzenie wsteczne jest wykorzystywane podczas analizy drzewa zależności. Oznacza to, że transakcje zostaną wycofane w kolejności odwrotnej do kolejności ich zatwierdzenia.

Tabela 39. Wszystkie możliwe wartości opcji wycofania.

Zasymulowano zależność klucza głównego poprzez usunięcie a następnie dodanie wiersza w tabeli *TACCOUNT* o dokładnie tym samym identyfikatorze. Wykorzystano w tym celu rekord o wartości kolumny klucza głównego równej 500. Aktualnie, prezentuje się on tak jak został przedstawiony w tabeli 40.

```
SELECT
  ACCOUNTID,
  EXTERNALID,
  USERNAME
FROM
  TACCOUNT
WHERE
  ACCOUNTID = 500;
```

ACCOUNTID	EXTERNALID	USERNAME
500	405176	Juniperus

Tabela 40. Zawartość rekordu przed modyfikacją.

Usunięto wiersz w jednej transakcji a następnie utworzono nowy o dokładnie tej samej wartości klucza głównego w obrębie nowej transakcji. Spowoduje to wystąpienie zależności klucza głównego pomiędzy dwiema transakcjami, zgodnie z opisem zawartym w tabeli 38.

```
DELETE
FROM
  TACCOUNT
WHERE
  ACCOUNTID = 500;
COMMIT;
INSERT
INTO
  TACCOUNT VALUES
  (
    500,
    500,
    500,
```

```
    SYSDATE,
    TO_DATE(NULL),
    1,
    NULL,
    12345,
    'Test',
    1,
    2,
    3
  );
COMMIT;
```

Po usunięciu i dodaniu, nowy wiersz o wartości klucza głównego 500 wygląda tak jak pokazano w tabeli 41.

```
SELECT
  ACCOUNTID,
  EXTERNALID,
  USERNAME
FROM
  TACCOUNT
WHERE
  ACCOUNTID = 500;
```

ACCOUNTID	EXTERNALID	USERNAME
500	12345	Test

Tabela 41. Zawartość rekordu po modyfikacji.

Aby zdobyć identyfikator transakcji, która usunęła wiersz, można skorzystać z zapytania zwracającego wszystkie wersje zdefiniowanego wiersza. Nie jest to jedyna możliwa droga pozyskania tegoż identyfikatora. Jest ona jednak najprostsza w realizacji, jeśli operacje były niedawno wykonane. Historia, zwracana przez zapytanie *Flashback Version Query*, została umieszczona w tabeli 42.

```
SELECT
  USERNAME,
  VERSIONS_STARTTIME STARTTIME,
  VERSIONS_XID XID,
  VERSIONS_OPERATION OPERATION
FROM
  TACCOUNT VERSIONS BETWEEN TIMESTAMP SYSTIMESTAMP - 1/24/8 AND
SYSTIMESTAMP
WHERE
  ACCOUNTID = 500;
```

USERNAME	STARTTIME	XID	OPERATION
Test	06-JAN-13 03.27.16.000000000 PM	200140076410000	I
Juniperus	06-JAN-13 03.27.16.000000000 PM	01001400E0300000	D

Tabela 42. Historia zmodyfikowanego rekordu.

Została podjęta próba wycofania transakcji, która usunęła wiersz. Skorzystano z opcji *NOCASCASE*. Wykonanie bloku kodu spowodowało wystąpienie błędu „*ORA-*

55504: Transaction conflicts in NOCASCADE mode". Jest to zachowanie takie, jakiego oczekiwano.

```
DECLARE
  VAXID SYS.XID_ARRAY;
BEGIN
  VAXID := SYS.XID_ARRAY('01001400E0300000');
  DBMS_FLASHBACK.TRANSACTION_BACKOUT(1, VAXID,
DBMS_FLASHBACK.NOCASCADE);
END;
```

Zgodnie z tabelą 39, należy wybrać opcję *CASCADE*, aby zależne transakcje zostały wycofane w odpowiedniej kolejności. Po wykonaniu poniższego bloku kodu, sprawdzono zawartość rekordu. Jest ona dokładnie taka sama jak w tabeli 40. Oznacza to, że dwie zależne transakcje zostały prawidłowo wycofane.

```
DECLARE
  VAXID SYS.XID_ARRAY;
BEGIN
  VAXID := SYS.XID_ARRAY('01001400E0300000');
  DBMS_FLASHBACK.TRANSACTION_BACKOUT(1, VAXID,
DBMS_FLASHBACK.CASCADE);
END;
```

Informację o wycofanych transakcjach można uzyskać poprzez wykonanie zapytań na widokach systemowych [*USER|DBA*]*_FLASHBACK_TXN_STATE* oraz [*USER|DBA*]*_FLASHBACK_TXN_REPORT* [19]. W pierwszym z nich znajdują się informacje o wycofanej transakcji, wszystkich transakcjach zależnych oraz identyfikator transakcji kompensującej. Z tabeli 43 wynika, że najpierw została wycofana transakcja zależna a potem dopiero transakcja, której identyfikator pojawił się wywołaniu procedury *TRANSACTION_BACKOUT* z pakietu *DBMS_FLASHBACK*.

```
SELECT
  COMPENSATING_XID,
  XID,
  DEPENDENT_XID
FROM
  DBA_FLASHBACK_TXN_STATE;
```

COMPENSATING_XID	XID	DEPENDENT_XID
02001F00EC0D0000	200140076410000	
02001F00EC0D0000	01001400E0300000	200140076410000

Tabela 43. Zależności pomiędzy wycofanymi transakcjami.

Drugi widok, [*USER|DBA*]*_FLASHBACK_TXN_REPORT*, przechowuje w formacie *XML* raport dotyczący transakcji kompensującej.

```
SELECT
  XID_REPORT
FROM
  DBA_FLASHBACK_TXN_REPORT
WHERE
  COMPENSATING_XID = '02001F00EC0D0000';
```

4.8.3. Flashback Data Archive

Mechanizm *Flashback Data Archive* umożliwia śledzenie oraz przechowywanie wszystkich transakcyjnych zmian zachodzących w tabeli poprzez cały okres jej istnienia [19]. Zwalnia to deweloperów z implementacji zewnętrznego mechanizmu. Konieczność przechowywania pełnej historii najczęściej jest podyktowana wymogami prawnymi. Wszelkie zmiany z reguły muszą być przechowywane przez zdefiniowany okres retencji. Aby skorzystać z mechanizmu muszą być spełnione następujące warunki [19]:

- Tabela nie może być zagnieżdżona, klastrowa, tymczasowa, zdalna lub też zewnętrzna.
- Tabela nie może zawierać kolumn typu *LONG* lub też bazujących na tabeli zagnieżdżonej.

Wymagane jest, aby archiwa były umieszczone w przestrzeni tabel, która ma załączoną opcję *Automatic Segment Space Management* (*ASSM*). Eliminuje ona konieczność definiowania parametrów alokacji dla przestrzeni tabel, takich jak *PCTUSED*, *Freelists* i *Freelist groups*. Od wersji *Oracle Database 10g Release 2*, każda nowa przestrzeń tabel jest domyślnie tworzona z wykorzystaniem *ASSM* [21]. Utworzono specjalną przestrzeń tabel, służącą do przechowywania archiwów tabel.

```
CREATE TABLESPACE OIMSARCHIVE DATAFILE
'/u02/app/oracle/oradata/testdbds/oimsarchive/oimsarchive01.dbf' SIZE
10M
  AUTOEXTEND ON;
```

Następnie w tejże przestrzeni tabel utworzono archiwum z okresem retencji równym jeden rok. Wiele tabel może korzystać z tego samego archiwum jednak, jak nazwa tegoż wskazuje, zostało ono utworzone tylko i wyłącznie dla historii tabeli *TACCOUNT*. Nie został zdefiniowany przydział przestrzeni. Z tego powodu archiwum będzie się rozrastać do nieograniczonych rozmiarów po to, aby zapewnić zdefiniowany okres retencji. Jeśli przydział przestrzeni zostanie zdefiniowany za pomocą klauzuli *QUOTA* to archiwum nie przekroczy określonego rozmiaru. Pojawia się jednak problem w przypadku, gdy archiwum jest za małe, aby zapewnić wymagany czas retencji. Jeśli rozmiar archiwum przekracza 90% dozwolonej wartości to generowane są ostrzeżenia.

Zignorowanie ich spowoduje, że w momencie osiągnięcia maksymalnego rozmiaru, operacje *DML*, staną się niemożliwe [10]. Problem dotyczy wszystkich tabel, które korzystają z zapełnionego archiwum.

```
CREATE FLASHBACK ARCHIVE FAACCOUNT TABLESPACE OIMSARCHIVE RETENTION 1
YEAR;
```

Jeśli wielkość archiwum osiągnęła próg ostrzeżenia lub maksymalny rozmiar, należy powiększyć przydział lub też przyciąć historię ręcznie. Poniższa instrukcja usunie dane historyczne starsze niż jeden dzień.

```
ALTER FLASHBACK ARCHIVE FAACCOUNT PURGE BEFORE TIMESTAMP (SYSTIMESTAMP
- INTERVAL '1' DAY);
```

Dla utworzonego archiwum, wydłużono okres retencji oraz ustawiono przydział w przestrzeni tabel *OIMSARCHIVE* na 5 GB.

```
ALTER FLASHBACK ARCHIVE FAACCOUNT MODIFY RETENTION 2 YEAR;
ALTER FLASHBACK ARCHIVE FAACCOUNT MODIFY TABLESPACE OIMSARCHIVE QUOTA
5G;
```

Domyślnie, mechanizm jest wyłączony dla wszystkich tabel. Zmodyfikowano tabelę *TACCOUNT*, tak, aby korzystała z utworzonego wcześniej archiwum. Archiwizacja tabeli może zostać włączona także na etapie tworzenia poprzez dodanie klauzuli *FLASHBACK ARCHIVE*.

```
ALTER TABLE TACCOUNT FLASHBACK ARCHIVE FAACCOUNT;
```

Dzięki archiwum, istnieje możliwość oglądania tabeli z dowolnego znacznika czasowego lub numeru *SCN*, który nie jest starszy niż jeden rok, ponieważ okres retencji archiwum wynosi właśnie tyle. Informacje o istniejących archiwach można uzyskać odpytując systemowy widok [*USER*|*DBA*]*_FLASHBACK_ARCHIVE*. Zawiera on podstawowe informacje o archiwach utworzonych w zakresie bazy danych lub schematu. Wybrane z nich znajdują się w tabeli 44.

```
SELECT
  FLASHBACK_ARCHIVE_NAME,
  RETENTION_IN_DAYS,
  CREATE_TIME
FROM
  DBA_FLASHBACK_ARCHIVE;
```

FLASHBACK_ARCHIVE_NAME	RETENTION_IN_DAYS	CREATE_TIME
FAACCOUNT	730	06-JAN-13 04.43.54.000000000 PM

Tabela 44. Podstawowe informacje o archiwum.

Aby dowiedzieć się o przydziałach dla istniejących archiwów, należy skorzystać z widoku [*USER*|*DBA*]_ *FLASHBACK_ARCHIVE_TS*. Tabela 45 pokazuje, że jest to 5 GB w przestrzeni tabel *OIMSARCHIVE*.

```
SELECT
  FLASHBACK_ARCHIVE_NAME,
  TABLESPACE_NAME,
  QUOTA_IN_MB
FROM
  DBA_FLASHBACK_ARCHIVE_TS;
```

FLASHBACK_ARCHIVE_NAME	TABLESPACE_NAME	QUOTA_IN_MB
FAACCOUNT	OIMSARCHIVE	5120

Tabela 45. Informacje o przydzielonej przestrzeni dyskowej dla archiwów.

Informacje o archiwizowanych tabelach znajdują się w widoku [*USER*|*DBA*]_ *_FLASHBACK_ARCHIVE_TABLES*. Dla każdej śledzonej tabeli, zostaje utworzona przez SZBD specjalna tabela archiwalna. W rozważanym przypadku jest to *SYS_FBA_HIST_73571*, tak jak prezentuje to tabela 46. Przechowuje ona wszystkie rekordy, które przestały być aktualne przez wykonanie operacji *DML*.

```
SELECT
  TABLE_NAME,
  FLASHBACK_ARCHIVE_NAME,
  ARCHIVE_TABLE_NAME
FROM
  DBA_FLASHBACK_ARCHIVE_TABLES;
```

TABLE_NAME	FLASHBACK_ARCHIVE_NAME	ARCHIVE_TABLE_NAME
TACCOUNT	FAACCOUNT	SYS_FBA_HIST_73571

Tabela 46. Informacje o archiwizacji tabel.

Korzystanie z archiwów ma też swoje skutki uboczne. Operacje *DDL*, modyfikujące archiwizowaną tabelę, są zabronione. Powodują one pojawienie się błędu „*ORA-55610: Invalid DDL statement on history-tracked table*” [19]. Pośród niedozwolonych operacji wymienia się:

- *ALTER TABLE* zmieniająca typ lub nazwę kolumny oraz partycjonująca,
- *DROP TABLE*,
- *RENAME TABLE*,
- *TRUNCATE TABLE*;

Aby zmodyfikować tabelę, należy wyłączyć śledzenie zmian. Spowoduje to, że cała zgromadzona historia zostanie utracona a tabela archiwum zostanie usunięta.

```
ALTER TABLE TACCOUNT NO FLASHBACK ARCHIVE;
```

5. Dostępność nowych funkcjonalności

W tabeli 47 znajduje się zestawienie wszystkich opisanych funkcjonalności wraz z numerem wersji, od której stały się one powszechnie dostępne.

Funkcjonalność	**Wersja**
Wirtualna kolumna	11.1
Ograniczenie klucza głównego i obcego dla kolumny wirtualnej	11.2
Operatory *PIVOT* i *UNPIVOT*	11.1
Usprawnienia w instrukcjach *DDL*	11.1
Zarządzanie statystykami	10.2
Partycjonowanie przedziałowe, referencyjne i systemowe	11.1
Nowe konfiguracje partycjonowania złożonego	11.1
Wyzwalacze złożone	11.1
Sterowanie porządkiem wykonania wyzwalaczy	11.1
Kompresja tabel dla baz transakcyjnych	11.1
Narzędzia *Flashback Database*, *Flashback Drop*, *Flashback Table*, *Flashback Version Query i Flashback Transaction Query*.	10.1
Narzędzia *Flashback Transaction* i *Flashback Data Archive*	11.1

Tabela 47. Dostępność nowych funkcjonalności.

6. Podsumowanie

W pracy zaprezentowano wybrane funkcjonalności dostępne w *Oracle Database 10g* i *11g*. Dla większości opisanych nowości, znaleziono rzeczywiste zastosowania w aplikacji testowej. Stały się one integralną częścią ostatecznej wersji projektu. Podczas implementacji mechanizmów natrafiono na problemy, które należało rozwiązać. Aplikacja postawiła przed autorem wyzwania, podobne do tych, jakie są stawiane architektom i deweloperom rozwiązań bazodanowych podczas codziennej pracy. Wielokrotnie, implementacja wymagała sięgnięcia po elementy, które są dostępne od wielu lat, ale ich znajomość jest konieczna do zrozumienia nowości, rozwiązania problemów i realizacji wyzwań. Pozwoliło to na jeszcze większe poszerzenie wiedzy o środowisku bazodanowym, niż spodziewano się.

Podczas wyboru zawartości pracy, kierowano się chęcią osiągnięcia lepszej skalowalności, wydajności i łatwości utrzymania aplikacji testowej. Niektóre funkcjonalności są szczególnie przydatne deweloperom, inne, administratorom baz danych. Nie zabrakło mechanizmów, których znajomość jest konieczna dla każdego, kto ma bezpośrednią styczność z bazami danych.

Kolumny wirtualne są bardzo dobrą alternatywą dla widoków rozszerzających podstawowy rezultat tabeli. Niskim nakładem pracy można dodać wartość obliczaną dynamicznie na podstawie wyrażenia. Ponadto, SZBD traktuje taką kolumnę jak zwykłą i umożliwia tworzenie dla niej indeksów oraz ograniczeń. Ukoronowaniem tejże funkcjonalności jest wykorzystywanie kolumny wirtualnej jako klucza partycjonowania. Niewątpliwie, jako wadę należy uznać fakt, że kluczem partycjonowania może stać się tylko kolumna wirtualna, której wyrażenie nie zawiera funkcji *PL/SQL*. Restrykcja dotyczy również funkcji, które utworzył użytkownik. Gdyby nie ta jedyna niedogodność, kolumna wirtualna sprawiłaby, że partycjonowanie systemowe stałoby się całkowicie bezużyteczne. Mogłaby ona w pełni realizować logikę niestandardowego klucza partycjonowania. Nie sprawdzono czy istnienie kolumny wirtualnej wpływa negatywnie na wydajność zapytań.

Operatory transformacji *PIVOT* i *UNPIVOT*, są funkcjonalnością, która ułatwia implementację oraz utrzymanie zapytań, pobierających dane do raportów. Obrót rezultatu relacyjnego względem określonych kolumn poprawia w ogromnym stopniu czytelność wyniku. Ponadto, oferują one zoptymalizowaną ścieżkę dostępu do danych, co

udowodniono poprzez testy wydajności. Jedyną zaobserwowaną wadą jest brak możliwości wprowadzania dynamicznych list elementów w klauzuli *IN*, w podstawowej wersji operacji *PIVOT*.

Usprawnienia w instrukcjach *DDL*, są bardzo łatwe w użyciu. Pomimo swojej prostoty, pozwalają na bezpieczniejsze, wydajniejsze i szybsze wprowadzanie zmian w środowisku produkcyjnym.

Zarządzanie statystykami pozwala uniknąć niespodzianek, jakie mogą wystąpić poza godzinami pracy. Nowo zebrane statystyki nie stają się od razu wiążące dla optymalizatora. Dzięki temu nie wystąpi nagły spadek wydajności związany z nieoptymalnymi planami wykonania zapytań. Zadanie gromadzące w nocy statystyki bazy danych zakończony swoje działanie i na tym poprzestanie. To administrator zadecyduje, kiedy nowe statystyki zaczną obowiązywać. W przypadku kontrolowanej publikacji i spadku wydajności, stare statystyki mogą zostać przywrócone w bardzo krótkim czasie.

Partycjonowanie pozwala na lepszą organizację zbiorów danych, co zaowocowało możliwością przechowywania wielokrotnie większej ilości danych, przy zachowaniu względnie stałej wydajności. W ręce projektantów, zostały oddane potężne mechanizmy. Udowodniono ich szczególną przydatność oraz skuteczność. Partycjonowanie przedziałowe jest bardzo prostym rozwinięciem partycjonowania zakresowego. Nowe konfiguracje partycjonowania złożonego są bez wątpienia cechą, którą należy brać pod uwagę podczas migracji bazy danych do nowej wersji SZBD. Zredukowano znacząco koszt pełnego odczytu, ponieważ dotyczy on tylko jednej podpartycji. Zwiększono tym samym wydajność systemu. Partycjonowanie referencyjne, pozwala na dziedziczenie schematu partycji. Wystarczy, aby dwie tabele znajdowały się w relacji rodzic-potomek. Łatwość utrzymania tabel znacząco się poprawia, ponieważ zmiany w partycjach rodzica są propagowane do tabeli potomka. Niestety, czas zaoszczędzony na etapie administracji zostanie pochłonięty przez strojenie zapytań *SQL* tak, aby prawidłowo korzystały z partycji tabeli potomka. Najmniej przydatną nowością jest partycjonowanie systemowe. Pozwala ono na zaimplementowanie zewnętrznego mechanizmu wyboru partycji, wtedy, gdy w tabeli nie istnieje jawny klucz partycjonowania. Docelowa partycja musi zostać zdefiniowana na etapie wstawiania rekordu do tabeli.

Wyzwalacze złożone pozwalają na utworzenie jednego obiektu, który reaguje na wszystkie możliwe zdarzenia, momenty i poziomy podczas operacji *DML*. Nie jest to szczególnie przydatna nowość, biorąc pod uwagę fakt, że ciężko utrzymać jeden wielki blok kodu, który rozpoczyna pracę przy wystąpieniu wszystkich możliwych sytuacji. Pomimo tego, każdy deweloper powinien samodzielnie ustosunkować się do tejże funkcjonalności i zadecydować czy warto z niej korzystać. Sterowanie porządkiem wykonania wyzwalaczy dla tego samego zdarzenia, momentu i poziomu, pozwala na łatwiejsze utrzymanie krótszych fragmentów kodu oraz zapewnia prawidłową realizację logiki biznesowej.

Kompresja tabeli stała się dostępna dla tradycyjnych systemów transakcyjnych. W znaczący sposób obniża koszt pełnego dostępu do tabeli poprzez mniejszą ilość bloków, które muszą zostać odczytane. Redukcja operacji *I/O* jest osiągana kosztem wyższego wykorzystania czasu procesora. Testy wydajności pokazują, że narzut nie jest duży. Każdy przypadek kompresji tabeli powinien być rozważany indywidualnie przez projektanta.

Technologia *Flashback* i narzędzia udostępnione przez nią, pozwalają na wycofanie tragicznych w skutkach błędnych operacji. Umożliwia powrót bazy danych lub jej wybranych fragmentów do świetności w ciągu kilku minut zamiast kilku godzin. *Flashback* to funkcjonalność, która jest obowiązkowa dla każdej osoby korzystającej bezpośrednio z bazy danych. Zasięg działania jest szeroki, ale bardzo krótki, ze względu na wysokie koszty utrzymania wymaganego czasu retencji. W przypadku prawdziwej błędnej operacji, nie będzie czasu na naukę. Biegła znajomość decyduje o tym czy operator zdąży skorzystać z *Flashback*. Prawidłowa, szybka i zdecydowana reakcja może sprawić, że końcowy użytkownik oprogramowania, korzystającego z bazy danych, nie odczuje w żaden sposób defektów.

Niniejsza praca nie wyczerpuje zagadnień, jakie należałoby poruszyć podczas rozważania nowych funkcjonalności dostępnych w *Oracle Database 10g* i *11g*. Nie przedstawiono wszystkich możliwych opcji dla opisanych mechanizmów. Niektóre z nich są tak rozbudowane, że chęć przetestowania wszystkich scenariuszy i dostępnych opcji, wymagałaby napisania o każdym z osoba więcej niż wynosi objętość całej pracy.

Podczas zgłębiania tajników najnowszych systemów bazodanowych firmy *Oracle*, szczególnie należy zwrócić uwagę na poniższe funkcjonalności, dla których zabrakło miejsca w tejże pracy:

- *Automatic Storage Management*,
- Przepisywanie zapytań,
- *SQL Plan Management*,
- Statystyki rozszerzone,
- Szyfrowanie przestrzeni tabel,
- *SQL Query Result Cache*;

7. Literatura

[1] W. Andrzejewski, Z. Królikowski i T. Morzy, „Bazy danych i systemy informatyczne oraz ich wpływ na rozwój informatyki w Polsce," w *Polskie i światowe osiągnięcia nauki: Nauki techniczne*, Gliwice, 2010.

[2] Oracle Corporation, Oracle 8i Concepts Release 2 (8.1.6), L. Lefty, D. Rehfield i C. Baird, Redaktorzy, 1999.

[3] Oracle Corporation, Oracle Database New Features Guide, 10g Release 1 (10.1), 2003.

[4] Oracle Corporation, Oracle Database New Features Guide, 10g Release 2 (10.2), 2008.

[5] Oracle Corporation, Oracle Database New Features Guide, 11g Release 1 (11.1), 2012.

[6] Oracle Corporation, Oracle Database New Features Guide, 11g Release 2 (11.2), 2011.

[7] R. G. Freeman i A. Nanda, Oracle Database 11g. Nowe możliwości, Helion, 2009.

[8] W. Edward, Oracle Database 10g. Administracja bazy danych w Linuksie, Helion, 2007.

[9] Oracle Corporation, Oracle Database Installation Guide, 11g Release 2 (11.2) for Linux, P. Jashnani, Red., 2012.

[10] Oracle Corporation, SQL Language Reference 11g Release 2 (11.2), D. Lorentz i M. B. Roeser, Redaktorzy, 2012.

[11] Oracle Corporation, Oracle Database Concepts, 10g Release 2 (10.2), M. Cyran, Red., 2005.

[12] Oracle Corporation, Oracle Database Administrator's Guide, 11g Release 1 (11.1), S. Fogel, Red., 2008.

[13] Oracle Corporation, Oracle Database Reference, 11g Release 2 (11.2), B. Rich, Red., 2012.

[14] Oracle Corporation, Oracle Database PL/SQL Packages and Types Reference, 11g Release 1 (11.1.), D. Raphaely, Red., 2008.

[15] Oracle Corporation, Oracle Database Administrator’s Guide, 10g Release 2 (10.2), S. Fogel, Red., 2006.

[16] Oracle Corporation, Oracle Database VLDB and Partitioning Guide, 11, T. Morales, Red., 2007.

[17] Oracle Corporation, Oracle9i Database Concepts, Release 2 (9.2), M. Cyran, Red., 2002.

[18] Oracle Corporation, Oracle9i Database Performance Tuning Guide and Reference, Release 2 (9.2), C. . D. Green, Red., 2002.

[19] Oracle Corporation, Oracle Database Advanced Application Developer's Guide, 11g Release 2 (11.2), S. Moore, Red., 2012.

[20] Oracle Corporation, Oracle Database Backup and Recovery User's Guide, 11g Release 2 (11.2), L. Ashdown, Red., 2011.

[21] B. Bryla i K. Loney, Oracle Database 11g. Podręcznik administratora baz danych, Helion, 2010.

8. Spis tabel

Tabela 1. Rozkład woluminów na wirtualnym dysku 9

Tabela 2. Specyfikacja rzeczywistej maszyny hosta. 10

Tabela 3. Przebieg instalacji oprogramowania bazy danych. 13

Tabela 4. Przebieg procedury tworzenia *listenera*. 14

Tabela 5. Przebieg procedury tworzenia bazy danych. 16

Tabela 6. Tabele rzadko rozbudowywane i aktualizowane. 25

Tabela 7. Tabele ciągle rozbudowywane i aktualizowane. 26

Tabela 8. Tabele ciągle rozbudowywane. 26

Tabela 9. Tabele raportowe. 26

Tabela 10. Ilości rekordów w tabelach po załadowaniu danych. 27

Tabela 11. Stany punktowe przed zastosowaniem operacji PIVOT. 32

Tabela 12. Stany punktowe po zastosowaniu operacji PIVOT. 33

Tabela 13. Porównanie wydajności dwóch zapytań, bez i z operatorem *PIVOT*. 35

Tabela 14. Rezultat operatora *PIVOT* z opcją *XML*. 36

Tabela 15. Stany punktowe po zastosowaniu *UNPIVOT* z opcją *INCLUDE NULLS*. 37

Tabela 16. Porównanie wydajności dwóch zapytań, bez i z operatorem *UNPIVOT*. 38

Tabela 17. Zachowanie poleceń *DDL* dla różnych wartości *DDL_LOCK_TIMEOUT*. 40

Tabela 18. Widoki słownikowe przechowujące oczekujące statystyki 44

Tabela 19. Procedury służące do przywracania statystyk. 47

Tabela 20. Widok partycji zakresowych dla tabeli *TPOINTSTATE*. 52

Tabela 21. Rozkład stanów punktowych ze względu na poziom. 54

Tabela 22. Rozkład stanów punktowych ze względu na typ. 54

Tabela 23. Widok partycji złożonych dla tabeli *TPOINTSTATE*. 56

Tabela 24. Widok podpartycji dla tabeli *TPOINTSTATE*. 57

Tabela 25. Widok partycji dla tabel powiązanych ograniczeniami. 59

Tabela 26. Informacje o partycjonowaniu, uzyskane ze słownika systemowego. 60
Tabela 27. Widok partycji dla tabel powiązanych ograniczeniami po modyfikacji. 60
Tabela 28. Dostępne opcje kompresji tabel. 67
Tabela 29. Obszar zaalokowany (E) dla tabeli skompresowanej i nieskompresowanej. 68
Tabela 30. Obszar zaalokowany (B) dla tabeli skompresowanej i nieskompresowanej. 68
Tabela 31. Parametry instancji bazy danych wpływające na narzędzie *SQL TRACE*. ... 69
Tabela 32. Porównanie wydajności operacji dla tabeli z kompresją i bez kompresji. ... 71
Tabela 33. Wybrane parametry inkarnacji bazy danych. 76
Tabela 34. Zawartość kosza po dwukrotnym usunięciu tabeli o tej samej nazwie. 77
Tabela 35. Pseudokolumny mechanizmu *Flashback Version Query*. 79
Tabela 36. Rezultat zastosowania *Flashback Version Query* dla zmienionego wiersza. 79
Tabela 37. Atomowe operacje wykonane w obrębie analizowanej transakcji. 80
Tabela 38. Zależności wykrywane przez *Flashback Transaction*. 81
Tabela 39. Wszystkie możliwe wartości opcji wycofania. 82
Tabela 40. Zawartość rekordu przed modyfikacją. 82
Tabela 41. Zawartość rekordu po modyfikacji. 83
Tabela 42. Historia zmodyfikowanego rekordu. 83
Tabela 43. Zależności pomiędzy wycofanymi transakcjami. 84
Tabela 44. Podstawowe informacje o archiwum. 86
Tabela 45. Informacje o przydzielonej przestrzeni dyskowej dla archiwów. 87
Tabela 46. Informacje o archiwizacji tabel. 87
Tabela 47. Dostępność nowych funkcjonalności. 88

9. Spis rysunków

Rysunek 1. Przepływ informacji pomiędzy modułami aplikacji. 19

Rysunek 2. Schemat testowej bazy danych 24

Rysunek 3. Plan wykonania zapytania dla widocznego indeksu. 42

Rysunek 4. Plan wykonania zapytania dla niewidocznego indeksu. 42

Rysunek 5. Plan wykonania zapytania przed partycjonowaniem. 49

Rysunek 6. Plan wykonania zapytania po partycjonowaniu zakresowym. 51

Rysunek 7. Plan wykonania zapytania po partycjonowaniu przedziałowo-listowym. ... 56

Rysunek 8. Wizualizacja inkarnacji bazy danych po przywróceniu. 75

Rysunek 9. Wizualizacja pełnej historii inkarnacji bazy danych. 76